第9章　Swing用户界面组件

- ▲ Swing与模型-视图-控制器设计模式
- ▲ 布局管理概述
- ▲ 文本输入
- ▲ 选择组件
- ▲ 菜单
- ▲ 复杂的布局管理
- ▲ 对话框

上一章主要介绍了如何使用Java中的事件模式。通过学习读者已经初步知道了构造图形用户界面的基本方法。本章将介绍构造功能更加齐全的图形用户界面所需要的一些重要工具。

下面，首先介绍Swing的基本体系结构。要想弄清如何有效地使用一些更高级的组件，必须了解底层的东西。然后，再讲述Swing中各种常用的用户界面组件，如文本框、单选按钮以及菜单等等。接下来，介绍在不考虑特定的用户界面观感时，如何使用Java中的布局管理器排列在窗口中的这些组件。最后，介绍如何在Swing中实现对话框。

本章囊括了基本的Swing组件，如文本组件、按钮和滑块等，这些都是基本的用户界面组件，使用十分频繁。Swing中的高级组件将在卷II中讨论。

9.1　Swing和模型 - 视图 - 控制器设计模式

前面说过，本章将从Swing组件的体系结构开始。首先，我们讨论一下设计模式的概念，然后再看一下Swing框架中最具影响力的“ 模型-视图-控制器 ”模式。

9.1.1　设计模式

在解决一个问题时，不需要从头做起，而是借鉴过去的经验，或者向做过相关工作的专家请教。设计模式就是一种方法，这种方法以一种结构化的方式展示专家们的心血。

近几年来，软件工程师们开始对这些模式进行汇总分类。这个领域的先驱者的灵感来源于建筑师Christopher Alexander的设计模式。他在《The Timeless Way of Building》（1979年，牛津大学出版）一书中，为公共和私人居住空间的建筑设计模式进行了分类。下面是一个典型的例子：

窗户位置

每个人都喜欢靠窗户的座位，可以画上凸出去的窗户、低窗台的大窗户以及放在这里的舒适椅子。如果一个房间中没有这样一个地方，很少有人会感到舒服和安逸。

如果房间中没有像这样“ 位置 ”的一个窗户，房间里的人就有可能要做出下列抉择：（1）舒适地坐下；（2）要充足的阳光。

显然，舒适的地方是房间中最想坐的地方，但它们远离窗户，这种冲突不可避免。

因此，对于白天长时间逗留的房间，至少要将一个窗户开在“ 窗户位置 ”处。

在Alexander的模式分类和软件模式的分类中，每种模式都遵循一种特定的格式。这些模式首先描述背景，即引发设计问题的情形；接着解释问题，通常这里会有几个冲突的因素；最终，权衡这些冲突，给出问题的解决方案。

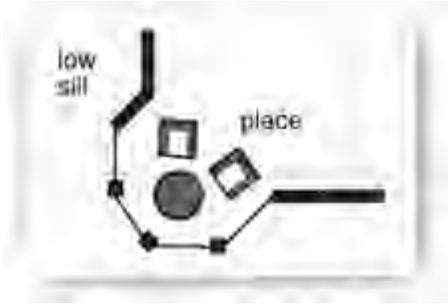


图9-1 窗户位置

在“窗户位置”模式中，背景是在白天逗留时间较长的房间。冲突因素就是既想舒适地坐下，又想拥有充足的光线。解决方案是找到一个“窗户位置”。

在“模型-视图-控制器”模式中，背景是显示信息和接收用户输入的用户界面系统。有关“模型-视图-控制器”模式将在接下来的章节中讲述。这里有几个冲突因素。对于同一数据来说，可能需要同时更新多个可视化表示。例如，为了适应各种观感标准，可能需要改变可视化表示形式；又例如，为了支持语音命令，可能需要改变交互机制。解决方案是将这些功能分布到三个独立的交互组件：模型、视图和控制器。

模型-视图-控制器模式并不是AWT和Swing设计中使用的惟一模式。下列是应用的另外几种模式：

- 容器和组件是“组合（composite）”模式
- 带滚动条的面板是“装饰（decorator）”模式
- 布局管理器是“策略（strategy）”模式

设计模式的另外一个最重要的特点是它们已经成为文化的一部分。只要谈论起模型-视图-控制器或“装饰”模式，遍及世界各地的程序员就会明白。因此，模式已经成为探讨设计方案的一种有效方法。

读者可以在Erich Gamma等编著的《Design Patterns —— Elements of Reusable Object-Oriented Software》（Addison-Wesley出版社，1995年出版[⊖]）一书中找到大量的实用软件模式的规范描述，这是一本研究模式运动的书籍。这里再强烈地推荐一本由Frank Buschmann等编著的《A System of Patterns》，John Wiley & Sons出版社于1996出版。这是一本不错的书籍，相对前一本，这本书更容易读懂。

9.1.2 模型-视图-控制器模式

让我们稍稍停顿一会儿，回想一下构成用户界面组件的各个组成部分，例如，按钮、复选框、文本框或者复杂的树形组件等。每个组件都有三个要素：

- 内容，如：按钮的状态（是否按下），或者文本框的文本。
- 外观（颜色，大小等等）。
- 行为（对事件的反应）。

这三个要素之间的关系是相当复杂的，即使对于最简单的组件（如：按钮）来说也是如此。很明显，按钮的外观显示取决于它的观感。Metal按钮的外观与Windows按钮或者Motif按钮的外观就不一样。另外，外观显示还要取决于按钮的状态：当按钮被按下时，按钮需要被重新绘制成

⊖ 本书中文版《设计模式-可复用面向对象软件的基础》已由机械工业出版社出版。——编辑注

另一种不同的外观。而状态取决于按钮接收到的事件。当用户在按钮上点击时，按钮就被按下。

当然，在程序中使用按钮时，只需要简单地把它看成是一个按钮，而不需要考虑它的内部工作和特性。毕竟，这些是实现按钮的程序员的工作。无论怎样，实现按钮的程序员就要对这些按钮考虑得细致一些了。毕竟，无论观感如何，他们必须实现这些按钮和其他用户界面组件，以便让这些组件正常地工作。

为了实现这样的需求，Swing设计者采用了一种很有名的设计模式（design pattern）：模型 - 视图 - 控制器（model-view-controller）模式。这种设计模式同其他许多设计模式一样，都遵循第5章介绍过的面向对象设计中的一个基本原则：限制一个对象拥有的功能数量。不要用一个按钮类完成所有的事情，而是应该让一个对象负责组件的观感，另一个对象负责存储内容。模型-视图-控制器 （MVC）模式告诉我们如何实现这种设计，实现三个独立的类：

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

这个模式明确地规定了三个对象如何进行交互。模型存储内容，它没有用户界面。按钮的内容非常简单，只有几个用来表示当前按钮是否按下，是否处于活动状态的标志等。文本框内容稍稍复杂一些，它是保存当前文本的字符串对象。这与视图显示的内容并不一致——如果内容的长度大于文本框的显示长度，用户就只能看到文本框可以显示的那一部分，如图9-2所示。

图9-2 文本框的模型和视图

模型必须实现改变内容和查找内容的方法。例如，一个文本模型中的方法有：在当前文本中添加或者删除字符以及把当前文本作为一个字符串返回等。记住：模型是完全不可见的。显示存储在模型中的数据是视图的工作。

注释：“ 模式 ” 这个术语可能不太贴切，因为人们通常把模式视为一个抽象概念的具体表示。汽车和飞机的设计者构造模式来模拟真实的汽车和飞机。但这种类比可能会使你对模型-视图-控制器模式产生错误的理解。在设计模式中，模型存储完整的内容，视图给出了内容的可视化显示（完整或者不完整）。一个更恰当的比喻应当是模特为画家摆好姿势。此时，就要看画家如何看待模特，并由此来作一张画了。那张画是一幅规矩的肖像画，或是一幅印象派作品，还是一幅立体派作品（以古怪的曲线来描绘四肢）完全取决于画家。

模型 - 视图 - 控制器模式的一个优点是一个模型可以有多个视图，其中每个视图可以显示全部内容的不同部分或不同形式。例如，一个HTML编辑器常常为同一内容在同一时刻提供两个视图：一个WYSIWYG（所见即所得）视图和一个 “ 原始标记 ” 视图（见图9-3）。当通过某一个视图的控制器对模型进行更新时，模式会把这种改变通知给两个视图。视图得到通知以后

就会自动地刷新。当然，对于一个简单的用户界面组件来说，如按钮，不需要为同一模型提供多个视图。

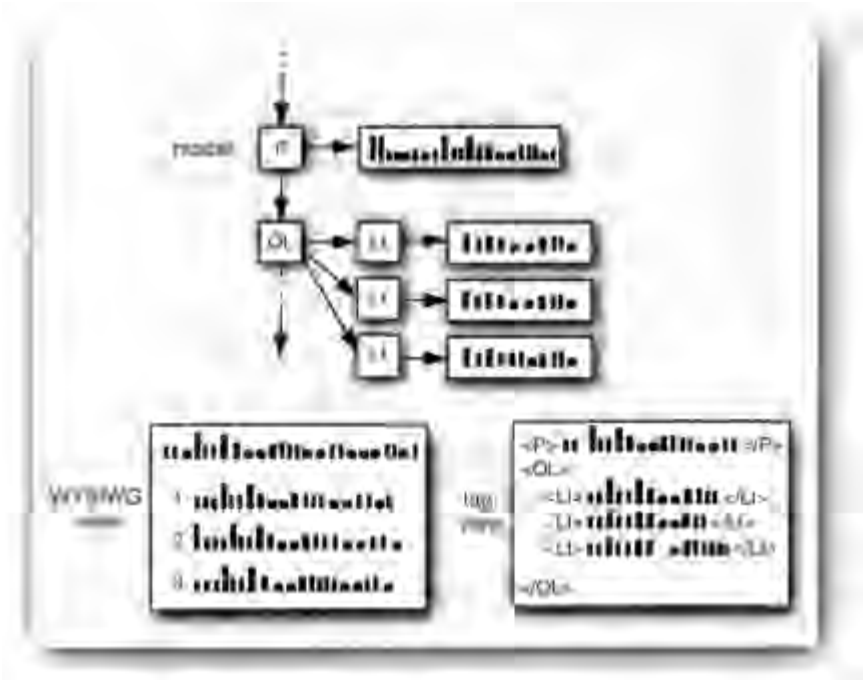


图9-3 同一模型的两个不同视图

控制器负责处理用户输入事件，如点击鼠标和敲击键盘。然后决定是否把这些事件转化成对模型或视图的改变。例如，如果用户在一个文本框中按下了一个字符键，控制器调用模型中的“插入字符”命令，然后模型告诉视图进行更新，而视图永远不会知道文本为什么改变了。但是如果用户按下了一个光标键，那么控制器会通知视图进行卷屏。卷动视图对实际文本不会有任何影响，因此模型永远不会知道这个事件的发生。

图9-4给出了模型、视图和控制器对象之间的交互。

在程序员使用Swing组件时，通常不需要考虑模型 - 视图 - 控制器体系结构。每个用户界面元素都有一个包装器类（如JButton或JTextField）来保存模型和视图。当需要查询内容（如文本域中的文本）时，包装器类会向模型询问并且返回所要的结果。当想改变视图时（例如，在一个文本域中移动光标位置），包装器类会把此请求转发给视图。然而，有时候包装器转发命令并不得力。在这种情况下，就必须直接地与模型打交道（不必直接操作视图——这是观感代码的任务）。

除了“本职工作”外，模型-视图-控制器模式吸引Swing设计者的主要原因是这种模式允许实现可插观感。每个按钮或者文本域的模型是独立于观感的。当然可视化表示完全依赖于特殊观感的用户界面设计，且控制器可以改变它。例如，在一个语音控制设备中，控制器需要处理的各种事件与使用键盘和鼠标的标准计算机完全不同。通过把底层模型与用户界面分离开，Swing设计者就能够重用模型的代码，甚至在程序运行时对观感进行切换。

当然，模式只能作为一种指导性的建议而并没有严格的戒律。没有一种模式能够适用于所

有情况。例如，使用“窗户位置”模式（设计模式中并非主要成分）来安排小卧室就不太合适。同样地，Swing设计者发现对于可插观感实现来说，使用模型-视图-控制器模式并非都是完美的。模型容易分离开，每个用户界面组件都有一个模型类。但是，视图和控制器的职责分工有时就不很明显，这样将会导致产生很多不同的类。当然，作为这些类的使用者来说，不必为这些细节费心。前面已经说过，这些类的使用者根本无需为模型操心，仅使用组件包装器类即可。

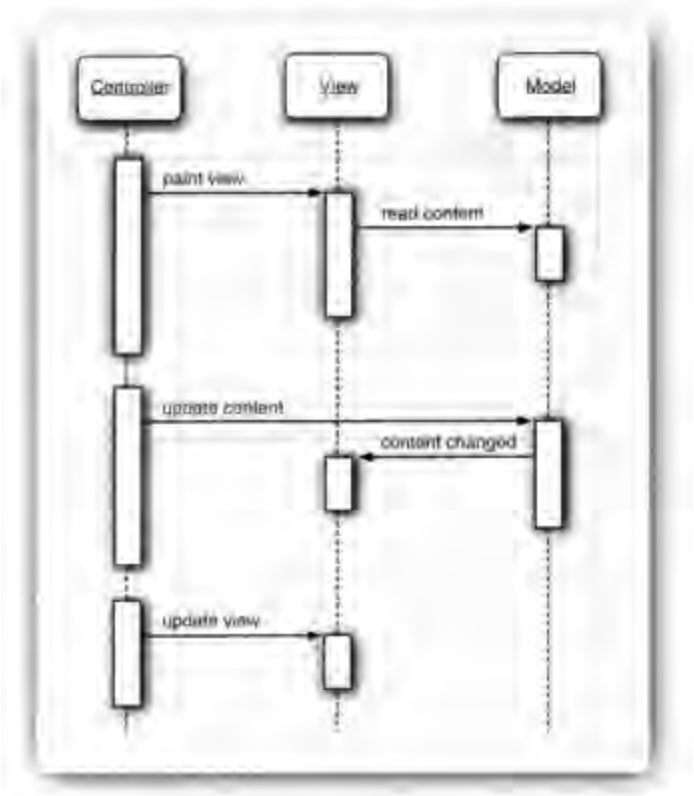


图9-4 给出了模型、视图、控制器对象之间的交互。

9.1.3 Swing按钮的模型 - 视图 - 控制器分析

前一章已经介绍了如何使用按钮，当时没有考虑模型、视图和控制器。按钮是最简单的用户界面元素，所以我们从按钮开始学习模型 - 视图 - 控制器模式会感觉容易些。对于更复杂的Swing组件来说，所遇到的类和接口都是类似的。

对于大多数组件来说，模型类将实现一个名字以Model结尾的接口，例如，按钮就实现了ButtonModel接口。实现了此接口的类可以定义各种按钮的状态。实际上，按钮并不复杂，在Swing库中有一个名为DefaultButtonModel的类就实现了这个接口。

读者可以通过查看ButtonModel接口中的特征来了解按钮模型所维护的数据类别。表9-1列出了这些特征。

每个JButton对象都存储了一个按钮模型对象，可以用下列方式得到它的引用。

```
JButton button = new JButton("Blue");
ButtonModel model = button.getModel();
```

表9-1 ButtonModel接口的属性

属 性 名	值
ActionCommand	与按钮关联的动作命令字符串
Mnemonic	按钮的快捷键
Armed	如果按钮被按下且鼠标仍在按钮上则为true
Enabled	如果按钮是可选择的则为true
Pressed	如果按钮被按下且鼠标按键没有释放则为true
Rollover	如果鼠标在按钮之上则为true
Selected	如果按钮已经被选择（用于复选框和单选按钮）则为true

实际上，不必关注按钮状态的零散信息，只有绘制它的视图才对此感兴趣。诸如按钮是否可用这样的重要信息完全可以通过JButton类得到（当然，JButton类也通过向它的模型询问来获得这些信息）。

下面查看一下ButtonModel接口中不包含的信息。模型不存储按钮标签或者图标。对于一个按钮来说，仅凭模型无法知道它的外观（实际上，在有关单选按钮组的一节中将会看到，这种纯粹的设计会给程序员带来一些麻烦）。

需要注意的是，同样的模型（即DefaultButtonModel）可用于下压按钮、单选按钮、复选框、甚至是菜单项。当然，这些按钮都有各自不同的视图和控制器。当使用Metal观感时，JButton类用BasicButtonUI类作为其视图；用ButtonUIListener类作为其控制器。通常，每个Swing组件都有一个相关的后缀为UI的视图对象，但并不是所有的Swing组件都有专门的控制器对象。

在阅读JButton底层工作的简介之后可能会想到：JButton究竟是什么？事实上，它仅仅是一个继承了JComponent的包装器类，JComponent包含了一个DefaultButtonModel对象，一些视图数据（例如按钮标签和图标）和一个负责按钮视图的BasicButtonUI对象。

9.2 布局管理器概述

在讨论每个Swing组件（例如：文本域和单选按钮）之前，首先介绍一下如何把这些组件排列在一个框架内。与Visual Basic不同，由于在JDK中没有表单设计器，所以需要通过编写代码来定制（布局）用户界面组件所在的位置。

当然，如果有支持Java的开发环境，就可能有某种布局工具来部分自动地或全部自动地完成这些布局任务。然而，弄清底层的实现方式是非常重要的，因为即使最好的工具有时也需要手工编码。

回顾上一章的程序，我们设计了几个按钮，点击这些按钮可以改变框架的背景颜色。所图9-5所示。

图9-5 包含三个按钮的面板

这几个按钮被放置在一个JPanel对象中，且用流布局管理器（flow layout manager）管理，这是面板的默认布局管理器。图9-6展示了向面板中添加多个按钮后的效果。正如读者所看到的，当一行的空间不够时，会将显示在新的一行上。

另外，按钮总是位于面板的中央，即使用户对框架进行缩放也是如此。如图9-7所示。

图9-6 用流布局管理六个按钮的面板

图9-7 改变面板尺寸后自动重新安排按钮

通常，组件放置在容器中，布局管理器决定容器中的组件具体放置的位置和大小。

按钮、文本域和其他的用户界面元素都继承于Component类，组件可以放置在面板这样的容器中。由于Container类继承于Component类，所以容器也可以放置在另一个容器中。图9-8给出了Component的类层次结构。

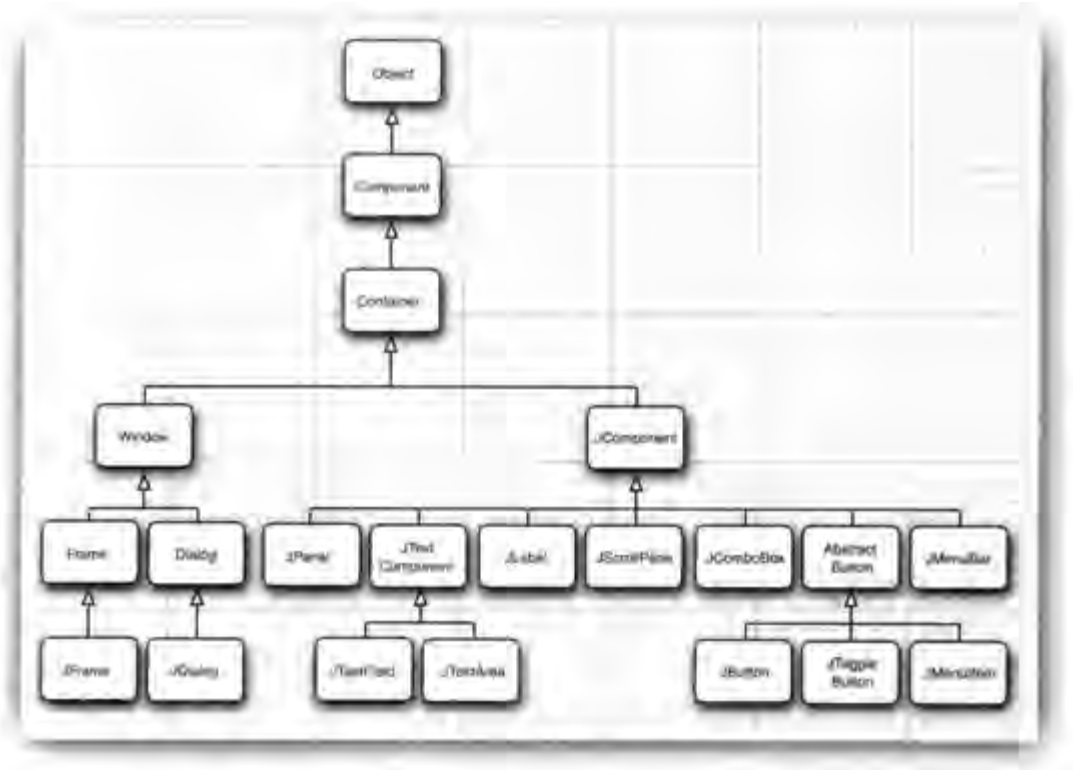


图9-8 Component的类层次结构

注释：可惜的是，继承层次有两点显得有点混乱。首先，像JFrame这样的顶层窗口是Container的子类，所以也是Component的子类，但却不能放在其他容器内。另外，JComponent是Container的子类，但不直接继承Component，因此，可以将其他组件添置到JButton中。（但无论如何，这些组件无法显示出来）。

每个容器都有一个默认的布局管理器，但可以重新进行设置。例如，使用下列语句：

```
panel.setLayout(new GridLayout(4, 4));
```

这个面板将用GridLayout类布局组件。可以往容器中添加组件。容器的add方法将把组件和放置的方位传递给布局管理器。

API `java.awt.Container 1.0`

- `Void SetLayout (LayoutManager m)`
  为容器设置布局管理器
- `Component add(Component c)`
- `Component add(Component c, Object constraints) 1.1`
  将组件添加到容器中，并返回组件的引用。
  参数：c　　要添加的组件
  　　constraints　　布局管理器理解的标识符

API `java.awt.FlowLayout 1.0`

- `FlowLayout ()`
- `FlowLayout (int align)`
- `FlowLayout (int align, int hgap, int vgap)`
  构造一个新的FlowLayout对象。
  参数：align　　LEFT、CENTER或者RIGHT
  　　hgap　　以像素为单位的水平间距（如果为负值，则强行重叠）
  　　vgap　　以像素为单位的垂直间距（如果为负值，则强行重叠）

9.2.1 边框布局

边框布局管理器（border layout manager）是每个JFrame的内容窗格的默认布局管理器。流布局管理器完全控制每个组件的放置位置，边框布局管理器则不然，它允许为每个组件选择一个放置位置。可以选择把组件放在内容窗格的中部、北部、南部、东部或者西部。如图9-9所示。

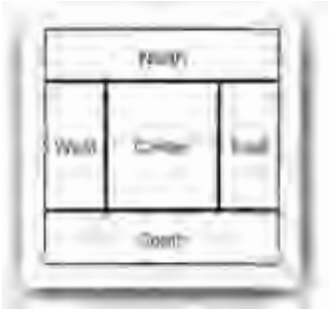


图9-9　边框布局

例如：

```
frame.add(component, BorderLayout.SOUTH);
```

先放置边缘组件，剩余的可用空间由中间组件占据。当容器被缩放时，边缘组件的厚度不会改变，而中部组件的大小会发生变化。在添加组件时可以指定BorderLayout类中的CENTER、NORTH、SOUTH、EAST和WEST常量。并非需要占用所有的位置，如果没有提供任何值，系统默认为CENTER。

✔ 注释：BorderLayout常量定义为字符串。例如：BorderLayout.SOUTH定义为字符串“SOUTH”。很多程序员喜欢直接使用字符串，因为这些字符串比较简短，例如，frame.add (component,“SOUTH”)。然而，如果字符串拼写有误，编译器不会捕获错误。

与流布局不同，边框布局会扩展所有组件的尺寸以便填满可用空间（流布局将维持每个组件的最佳尺寸）。当将一个按钮添加到容器中时会出现问题：

```
frame.add(yellowButton, BorderLayout.SOUTH); // don't
```

图9-10给出了执行上述语句的显示效果。按钮扩展至填满框架的整个南部区域。而且，如果再将另外一个按钮添加到南部区域，就会取代第一个按钮。

解决这个问题的常见方法是使用另外一个面板（panel）。例如，如图9-11所示。屏幕底部的三个按钮全部包含在一个面板中。这个面板被放置在内容窗格的南部。

图9-10 边框布局管理一个按钮

图9-11 面板放置在框架的南部区域

要想得到这种效果，首先需要创建一个新的JPanel对象，然后逐一将按钮添加到面板中。面板的默认布局管理器是FlowLayout，这恰好符合我们的需求。随后使用在前面已经看到的add方法将每个按钮添加到面板中。每个按钮的放置位置和尺寸完全处于FlowLayout布局管理器的控制之下。这意味着这些按钮将置于面板的中央，并且不会扩展至填满整个面板区域。最后，将这个面板添加到框架的内容窗格中。

```
JPanel panel = new JPanel();
panel.add(yellowButton);
panel.add(blueButton);
panel.add(redButton);
frame.add(panel, BorderLayout.SOUTH);
```

边框布局管理器将会扩展面板大小，直至填满整个南部区域。

API `java.awt.BorderLayout 1.0`

- `BorderLayout()`
- `BorderLayout( int hgap, int vgap)`

构造一个新的BorderLayout对象。

参数：hgap 以像素为单位的水平间距（如果为负值，则强行重叠）

vgap 以像素为单位的垂直间距（如果为负值，则强行重叠）

9.2.2 网格布局

网格布局像电子数据表一样，按行列排列所有的组件。不过，它的每个单元大小都是一样的。图9-12显示的计算器程序就使用了网格布局来排列计算器按钮。当缩放窗口时，计算器按钮将随之变大或变小，但所有的按钮尺寸始终保持一致。

在网格布局对象的构造器中，需要指定行数和列数：

```
panel.setLayout(new GridLayout(5, 4));
```

添加组件，从第一行的第一列开始，然后是第一行的第二列，以此类推。

```
panel.add(new JButton("1"));
panel.add(new JButton("2"));
```

图9-12 计算器

例9-1是计算器程序的源代码。这是一个常规的计算器，而不像Java指南中所提到的“逆波兰”那样古怪。在这个程序中，在将组件添加到框架之后，调用了pack方法。这个方法用于将所有组件以最佳的高度和宽度显示在框架中。

当然，极少有像计算器这样整齐的布局。实际上，在组织窗口的布局时小网格（通常只有一行或者一列）比较有用。例如，如果想放置一行尺寸都一样的按钮，就可以将这些按钮放置在一个面板里，这个面板使用只有一行的网格布局进行管理。

例9-1 Calculator.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.33 2007-06-12
7.  * @author Cay Horstmann
8.  */
9. public class Calculator
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                CalculatorFrame frame = new CalculatorFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame with a calculator panel.
27.  */
28. class CalculatorFrame extends JFrame
29. {
30.    public CalculatorFrame()
31.    {
32.       setTitle("Calculator");
33.       CalculatorPanel panel = new CalculatorPanel();
34.       add(panel);
35.       pack();
36.    }
```

```
37. }
38.
39. /**
40.  * A panel with calculator buttons and a result display.
41.  */
42. class CalculatorPanel extends JPanel
43. {
44.    public CalculatorPanel()
45.    {
46.       setLayout(new BorderLayout());
47.
48.       result = 0;
49.       lastCommand = "=";
50.       start = true;
51.
52.       // add the display
53.
54.       display = new JButton("0");
55.       display.setEnabled(false);
56.       add(display, BorderLayout.NORTH);
57.
58.       ActionListener insert = new InsertAction();
59.       ActionListener command = new CommandAction();
60.
61.       // add the buttons in a 4 x 4 grid
62.
63.       panel = new JPanel();
64.       panel.setLayout(new GridLayout(4, 4));
65.
66.       addButton("7", insert);
67.       addButton("8", insert);
68.       addButton("9", insert);
69.       addButton("/", command);
70.
71.       addButton("4", insert);
72.       addButton("5", insert);
73.       addButton("6", insert);
74.       addButton("*", command);
75.
76.       addButton("1", insert);
77.       addButton("2", insert);
78.       addButton("3", insert);
79.       addButton("-", command);
80.
81.       addButton("0", insert);
82.       addButton(".", insert);
83.       addButton("=", command);
84.       addButton("+", command);
85.
86.       add(panel, BorderLayout.CENTER);
87.    }
88.
89.    /**
90.     * Adds a button to the center panel.
91.     * @param label the button label
92.     * @param listener the button listener
93.     */
```

```
94.    private void addButton(String label, ActionListener listener)
95.    {
96.       JButton button = new JButton(label);
97.       button.addActionListener(listener);
98.       panel.add(button);
99.    }
100.
101.   /**
102.    * This action inserts the button action string to the end of the display text.
103.    */
104.   private class InsertAction implements ActionListener
105.   {
106.      public void actionPerformed(ActionEvent event)
107.      {
108.         String input = event.getActionCommand();
109.         if (start)
110.         {
111.            display.setText("");
112.            start = false;
113.         }
114.         display.setText(display.getText() + input);
115.      }
116.   }
117.
118.   /**
119.    * This action executes the command that the button action string denotes.
120.    */
121.   private class CommandAction implements ActionListener
122.   {
123.      public void actionPerformed(ActionEvent event)
124.      {
125.         String command = event.getActionCommand();
126.
127.         if (start)
128.         {
129.            if (command.equals("-"))
130.            {
131.               display.setText(command);
132.               start = false;
133.            }
134.            else lastCommand = command;
135.         }
136.         else
137.         {
138.            calculate(Double.parseDouble(display.getText()));
139.            lastCommand = command;
140.            start = true;
141.         }
142.      }
143.   }
144.
145.   /**
146.    * Carries out the pending calculation.
147.    * @param x the value to be accumulated with the prior result.
148.    */
149.   public void calculate(double x)
150.   {
```

```
151.       if (lastCommand.equals("+")) result += x;
152.       else if (lastCommand.equals("-")) result -= x;
153.       else if (lastCommand.equals("*")) result *= x;
154.       else if (lastCommand.equals("/")) result /= x;
155.       else if (lastCommand.equals("=")) result = x;
156.       display.setText("" + result);
157.    }
158.
159.    private JButton display;
160.    private JPanel panel;
161.    private double result;
162.    private String lastCommand;
163.    private boolean start;
164. }
```

API java.awt.GridLayout 1.0

- GridLayout(int rows, int cols)
- GridLayout(int rows, int columns, int hgap, int vgap)

构造一个新的GridLayout对象。rows或者columns可以为零，但不能同时为零，指定的每行或每列的组件数量可以任意的。

参数：rows 网格的行数
columns 网格的列数
hgap 以像素为单位的水平间距（如果为负值，则强行重叠）
vgap 以像素为单位的垂直间距（如果为负值，则强行重叠）

API java.awt.Window 1.0

- void pack()

缩放窗口时，将组件调整至最佳尺寸。

9.3 文本输入

现在终于可以开始介绍Swing用户界面组件了。首先，介绍具有用户输入和编辑文本功能的组件。文本域（JTextField）和文本区（JTextArea）组件用于获取文本输入。文本域只能接收单行文本的输入，而文本区能够接收多行文本的输入。JPassword也只能接收单行文本的输入，但不会将输入的内容显示出来。

这三个类都继承于JTextComponent类。由于JTextComponent是一个抽象类，所以不能够构造这个类的对象。另外，在Java中常会看到这种情况。在查看API文档时，发现自己正在寻找的方法实际上来自于父类JTextComponent，而不是来自派生类自身。例如，在一个文本域和文本区内获取（get）、设置（set）文本的方法实际上都是JTextComponent类中的方法。

API javax.swing.text.JTextComponent 1.2

- String getText()

- `void setText(String text)`
  获取或设置文本组件中的文本。
- `boolean isEditable()`
- `void setEditable(boolean b)`
  获取或设置editable特性，这个特性决定了用户是否可以编辑文本组件中的内容。

9.3.1 文本域

把文本域添加到窗口的常用办法是将它添加到面板或者其他容器中，这与添加按钮完全一样：

```
JPanel panel = new JPanel();
JTextField textField = new JTextField("Default input", 20);
panel.add(textField);
```

这段代码将添加一个文本域，同时通过传递字符串“Default input”进行初始化。构造器的第二个参数设置了文本域的宽度。在这个示例中，宽度值为20“列”。但是，这里所说的列不是一个精确的测量单位。一列就是在当前使用的字体下一个字符的宽度。如果希望文本域最多能够输入n个字符，就应该把宽度设置为n列。在实际中，这样做效果并不理想，应该将最大输入长度再多设1~2个字符。列数只是给AWT设定首选（preferred）大小的一个提示。如果布局管理器需要缩放这个文本域，它会调整文本域的大小。在JTextField的构造器中设定的宽度并不是用户能输入的字符个数的上限。用户可以输入一个更长的字符串，但是当文本长度超过文本域长度时输入就会滚动。用户通常不喜欢滚动文本域，因此应该尽量把文本域设置的宽一些。如果需要在运行时重新设置列数，可以使用setColumns方法。

提示：使用setColumns方法改变了一个文本域的大小之后，需要调用包含这个文本框的容器的revalidate方法。

```
textField.setColumns(10);
panel.revalidate();
```

revalidate方法会重新计算容器内所有组件的大小，并且对它们重新进行布局。调用revalidate方法以后，布局管理器会重新设置容器的大小，然后就可以看到改变尺寸后的文本域了。

revalidate方法是JComponent类中的方法。它并不是马上就改变组件大小，而是给这个组件加一个需要改变大小的标记。这样就避免了多个组件改变大小时带来的重复计算。但是，如果想重新计算一个JFrame中的所有组件，就必须调用validate方法——JFrame没有扩展JComponent。

通常情况下，希望用户在文本域中键入文本（或者编辑已经存在的文本）。文本域一般初始为空白。只要不为JTextField构造器提供字符串参数，就可以构造一个空白文本域：

```
JTextField textField = new JTextField(20);
```

可以在任何时候调用setText方法改变文本域中的内容。这个方法是从前面提到的JTextComponent中继承而来的。例如：

```
textField.setText("Hello!");
```

并且，在前面已经提到，可以调用getText方法来获取用户键入的文本。这个方法返回用户输入的文本。如果想要将getText方法返回的文本域中的内容的前后空格去掉，就应该调用trim方法：

```
String text = textField.getText().trim();
```

如果想要改变显示文本的字体，就调用setFont方法。

API javax.swing.JTextField 1.2

- JTextField(int cols)
  构造一个给定列数的空JTextField对象。
- JTextField(String text, int cols)
  构造一个给定列数、给定初始字符串的JTextField对象。
- int getColumns()
- void setColumns(int cols)
  获取或设置文本域使用的列数。

API javax.swing.JComponent 1.2

- void revalidate()
  重新计算组件的位置和大小。
- void setFont(Font f)
  设置组件的字体。

API java.awt.Component 1.0

- void validate()
  重新计算组件的位置和大小。如果组件是容器，容器中包含的所有组件的位置和大小也被重新计算。
- Font getFont()
  获取组件的字体。

9.3.2 标签和标签组件

标签是容纳文本的组件，它们没有任何的修饰（例如没有边缘），也不能响应用户输入。可以利用标签标识组件。例如：与按钮不同，文本域没有标识它们的标签。要想用标识符标识这种不带标签的组件，应该

1）用相应的文本构造一个JLabel组件。

2）将标签组件放置在距离需要标识的组件足够近的地方，以便用户可以知道标签所标识的组件。

JLabel的构造器允许指定初始文本和图标，也可以选择内容的排列方式。可以用Swing

Constants接口中的常量来指定排列方式。在这个接口中定义了几个很有用的常量，如LEFT、RIGHT、CENTER、NORTH、EAST等。JLabel是实现这个接口的一个Swing类。因此，可以指定右对齐标签：

```
JLabel label = new JLabel("User name: ", SwingConstants.RIGHT);
```

或者

```
JLabel label = new JLabel("User name: ", JLabel.RIGHT);
```

利用setText和setIcon方法可以在运行期间设置标签的文本和图标。

提示：从JDK1.3开始，可以在按钮、标签和菜单项上使用无格式文本或HTML文本。我们不推荐在按钮上使用HTML文本——这样会影响观感。但是HTML文本在标签中是非常有效的。只要简单地将标签字符串放置在<html>...</html>中即可：

```
label = new JLabel("<html><b>Required</b> entry:</html>");
```

警告——包含HTML标签的第一个组件需要延迟一段时间才能显示出来，这是因为需要加载相当复杂的HTML显示代码。

与其他组件一样，标签也可以放置在容器中。这就是说，可以利用前面介绍的技巧将标签放置在任何需要的地方。

`javax.swing.JLabel 1.2`

- `JLabel(String text)`
- `JLabel(Icon icon)`
- `JLabel(String text, int align)`
- `JLabel(String text, Icon icon, int align)`

构造一个标签。

参数：text 标签中的文本
icon 标签中的图标
align 一个SwingConstants的常量LEFT（默认），CENTER或者RIGHT

- `String getText()`
- `void setText(String text)`

获取或设置标签的文本。

- `Icon getIcon()`
- `void setIcon(Icon icon)`

获取或设置标签的图标。

9.3.3 密码域

密码域是一种特殊类型的文本域。为了避免有某种企图的人看到密码，用户输入的字符不显示出来。每个输入的字符都用回显字符（echo character）表示，典型的回显字符是星号（*）。Swing提供了JPasswordField类来实现这样的文本域。

密码域是另一个应用模型-视图-控制器体系模式的例子。密码域采用与常规的文本域相同的模型来存储数据，但是，它的视图却改为显示回显字符，而不是实际的字符。

API javax.swing.JPasswordField 1.2

- `JPasswordField(String text, int columns)`
  构造一个新的密码域对象。
- `void setEchoChar(char echo)`
  为密码域设置回显字符。注意：独特的观感可以选择自己的回显字符。0表示重新设置为默认的回显字符。
- `char[ ] getPassword()`
  返回密码域中的文本。为了安全起见，在使用之后应该覆写返回的数组内容（密码并不是以String的形式返回，这是因为字符串在被垃圾回收器回收之前会一直驻留在虚拟机中）。

9.3.4 文本区

有时，用户的输入超过一行。正像前面提到的，需要使用JTextArea组件来接收这样的输入。当在程序中放置一个文本区组件时，用户就可以输入多行文本，并用ENTER键换行。每行都以一个“\n”结尾。图9-13显示了一个工作的文本区。

图9-13 文本组件

在JTextArea组件的构造器中，可以指定文本区的行数和列数。例如：

```
textArea = new JTextArea(8, 40); // 8 lines of 40 columns each
```

与文本域一样。出于稳妥的考虑，参数columns应该设置的大一些。另外，用户并不受限于输入指定的行数和列数。当输入过长时，文本会滚动。还可以用setColumns方法改变列数，用setRows方法改变行数。这些数值只是首选大小——布局管理器可能会对文本区进行缩放。

如果文本区的文本超出显示的范围，那么剩下的文本就会被剪裁掉。可以通过开启换行特性来避免裁剪过长的行：

```
textArea.setLineWrap(true); // long lines are wrapped
```

换行只是视觉效果；文档中的文本没有改变，在文本中并没有插入“\n”字符。

9.3.5 滚动窗格

在Swing中，文本区没有滚动条。如果需要滚动条，可以将文本区插入到滚动窗格（scroll pane）中。

```
textArea = new JTextArea(8, 40);
JScrollPane scrollPane = new JScrollPane(textArea);
```

现在滚动窗格管理文本区的视图。如果文本超出了文本区可以显示的范围，滚动条就会自动地出现，并且在删除部分文本后，当文本能够显示在文本区范围内时，滚动条会再次自动地消失。滚动是由滚动窗格内部处理的，编写程序时无需处理滚动事件。

这是一种为任意组件添加滚动功能的通用机制，而不是文本区特有的。也就是说，要想为组件添加滚动条，只需将它们放入一个滚动窗格中即可。

例9-2展示了各种文本组件。这个程序只是简单地显示了一个文本域、一个密码域和一个带滚动条的文本区。文本域和密码域都使用了标签。点击“ Insert ”会将组件中的内容插入到文本区中。

注释：JTextArea组件只显示无格式的文本，没有字体或者格式设置。如果想要显示格式化文本（如HTML或者RTF），就需要使用JEditorPane和JTextPane类。在卷II将详细地讨论这几个类。

例9-2 TextComponentTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.40 2007-04-27
7.  * @author Cay Horstmann
8.  */
9. public class TextComponentTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                TextComponentFrame frame = new TextComponentFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame with sample text components.
27.  */
28. class TextComponentFrame extends JFrame
29. {
30.    public TextComponentFrame()
31.    {
32.       setTitle("TextComponentTest");
33.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
34.
35.       final JTextField textField = new JTextField();
36.       final JPasswordField passwordField = new JPasswordField();
37.
38.       JPanel northPanel = new JPanel();
39.       northPanel.setLayout(new GridLayout(2, 2));
40.       northPanel.add(new JLabel("User name: ", SwingConstants.RIGHT));
41.       northPanel.add(textField);
```

```
42.       northPanel.add(new JLabel("Password: ", SwingConstants.RIGHT));
43.       northPanel.add(passwordField);
44.
45.       add(northPanel, BorderLayout.NORTH);
46.
47.       final JTextArea textArea = new JTextArea(8, 40);
48.       JScrollPane scrollPane = new JScrollPane(textArea);
49.
50.       add(scrollPane, BorderLayout.CENTER);
51.
52.       // add button to append text into the text area
53.
54.       JPanel southPanel = new JPanel();
55.
56.       JButton insertButton = new JButton("Insert");
57.       southPanel.add(insertButton);
58.       insertButton.addActionListener(new ActionListener()
59.          {
60.             public void actionPerformed(ActionEvent event)
61.             {
62.                textArea.append("User name: " + textField.getText() + " Password: "
63.                      + new String(passwordField.getPassword()) + "\n");
64.             }
65.          });
66.
67.       add(southPanel, BorderLayout.SOUTH);
68.
69.       // add a text area with scrollbars
70.
71.    }
72.
73.    public static final int DEFAULT_WIDTH = 300;
74.    public static final int DEFAULT_HEIGHT = 300;
75. }
```

API javax.swing.JTextArea 1.2

- JTextArea()
- JTextArea(int rows, int cols)
- JTextArea(String text, int rows, int cols)
  构造一个新的文本区对象。
- void setColumns(int cols)
  设置文本区应该使用的首选列数。
- void setRows(int rows)
  设置文本区应该使用的首选行数。
- void append(String newText)
  将给定的文本追加到文本区中已有文本的尾部。
- void setLineWrap(boolean wrap)
  打开或关闭换行。

- `void setWrapStyleWord(boolean word)`
  如果word是true，超长的行会在字边框处换行。如果为false，超长的行被截断而不考虑字边框。
- `void setTabSize(int c)`
  将制表符（tab stop）设置为c 列。注意，制表符不会被转化为空格，但可以让文本对齐到下一个制表符处。

API `javax.swing.JScrollPane 1.2`

- `JScrollPane(Component c)`
  创建一个滚动窗格，用来显示指定组件的内容。当组件内容超过显示范围时，滚动条会自动地出现。

9.4 选择组件

前面已经讲述了如何获取用户输入的文本。然而，在很多情况下，可能更加愿意给用户几种选项，而不让用户在文本组件中输入数据。使用一组按钮或者选项列表让用户做出选择（这样也免去了检查错误的麻烦）。在本节中，将介绍如何编写程序来实现复选框、单选按钮、选项列表以及滑块。

9.4.1 复选框

如果想要接收的输入只是“是”或“非”，就可以使用复选框组件。复选框自动地带有标识标签。用户通过点击某个复选框来选择相应的选项，再点击则取消选取。当复选框获得焦点时，用户也可以通过按空格键来切换选择。

图9-14 复选框

图9-14所示的程序中有两个复选框，其中一个用于打开或关闭字体倾斜属性，而另一个用于控制加粗属性。注意，第二个复选框有焦点，这一点可以由它周围的矩形框看出。只要用户点击某个复选框，程序就会刷新屏幕以便应用新的字体属性。

复选框需要一个紧邻它的标签来说明其用途。在构造器中指定标签文本。

```
bold = new JCheckBox("Bold");
```

可以使用setSelected方法来选定或取消选定复选框。例如：

```
bold.setSelected(true);
```

isSelected方法将返回每个复选框的当前状态。如果没有选取则为false，否则为true。

当用户点击复选框时将触发一个动作事件。通常，可以为复选框设置一个动作监听器。在下面程序中，两个复选框使用了同一个动作监听器。

```
ActionListener listener = . . .
bold.addActionListener(listener);
italic.addActionListener(listener);
```

actionPerformed方法查询bold和italic两个复选框的状态，并且把面板中的字体设置为常规、

加粗、倾斜或者粗斜体。

```
public void actionPerformed(ActionEvent event)
{
   int mode = 0;
   if (bold.isSelected()) mode += Font.BOLD;
   if (italic.isSelected()) mode += Font.ITALIC;
   label.setFont(new Font("Serif", mode, FONTSIZE));
}
```

例9-3给出了复选框例子的全部代码。

例9-3 CheckBoxTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.33 2007-06-12
7.  * @author Cay Horstmann
8.  */
9. public class CheckBoxTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                CheckBoxFrame frame = new CheckBoxFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame with a sample text label and check boxes for selecting font attributes.
27.  */
28. class CheckBoxFrame extends JFrame
29. {
30.    public CheckBoxFrame()
31.    {
32.       setTitle("CheckBoxTest");
33.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
34.
35.       // add the sample text label
36.
37.       label = new JLabel("The quick brown fox jumps over the lazy dog.");
38.       label.setFont(new Font("Serif", Font.PLAIN, FONTSIZE));
39.       add(label, BorderLayout.CENTER);
40.
41.       // this listener sets the font attribute of
42.       // the label to the check box state
43.
```

```
44.       ActionListener listener = new ActionListener()
45.          {
46.             public void actionPerformed(ActionEvent event)
47.             {
48.                int mode = 0;
49.                if (bold.isSelected()) mode += Font.BOLD;
50.                if (italic.isSelected()) mode += Font.ITALIC;
51.                label.setFont(new Font("Serif", mode, FONTSIZE));
52.             }
53.          };
54.
55.       // add the check boxes
56.
57.       JPanel buttonPanel = new JPanel();
58.
59.       bold = new JCheckBox("Bold");
60.       bold.addActionListener(listener);
61.       buttonPanel.add(bold);
62.
63.       italic = new JCheckBox("Italic");
64.       italic.addActionListener(listener);
65.       buttonPanel.add(italic);
66.
67.       add(buttonPanel, BorderLayout.SOUTH);
68.    }
69.
70.    public static final int DEFAULT_WIDTH = 300;
71.    public static final int DEFAULT_HEIGHT = 200;
72.
73.    private JLabel label;
74.    private JCheckBox bold;
75.    private JCheckBox italic;
76.
77.    private static final int FONTSIZE = 12;
78. }
```

API javax.swing.JCheckBox 1.2

- JCheckBox(String label)
- JCheckBox(String label, Icon icon)
  构造一个复选框，初始没有被选择。
- JCheckBox(String label, boolean state)
  用给定的标签和初始化状态构造一个复选框。
- boolean isSelected ()
- void setSelected(boolean state)
  获取或设置复选框的选择状态。

9.4.2 单选按钮

在前一个例子中，对于两个复选框，用户既可以选择一个、两个，也可以两个都不选。在

很多情况下，我们需要用户只选择几个选项当中的一个。当用户选择另一项的时候，前一项就自动地取消选择。这样一组选框通常称为单选按钮组（Radio Button Group），这是因为这些按钮的工作很像收音机上的电台选择按钮。当按下一个按钮时，前一个按下的按钮就会自动弹起。图9-15给出了一个典型的例子。这里允许用户在多个选择中选择字体的大小，即小、中、大和超大，但是，每次用户只能选择一个。

图9-15 单选按钮组

在Swing中，实现单选按钮组非常简单。为单选按钮组构造一个ButtonGroup的对象。然后，再将JRadioButton类型的对象添加到按钮组中。按钮组负责在新按钮被按下时，取消前一个被按下的按钮的选择状态。

```
ButtonGroup group = new ButtonGroup();

JRadioButton smallButton = new JRadioButton("Small", false);
group.add(smallButton);

JRadioButton mediumButton = new JRadioButton("Medium", true);
group.add(mediumButton);
. . .
```

构造器的第二个参数为true表明这个按钮初始状态是被选择，其他按钮构造器的这个参数为false。注意，按钮组仅仅控制按钮的行为，如果想把这些按钮组织在一起布局，需要把它们添加到容器中，如JPanel。

如果再看一下图9-14和图9-15则会发现，单选按钮与复选框的外观是不一样的。复选框为正方形，并且如果被选择，这个正方形中会出现一个对钩的符号。单选按钮是圆形，选择以后圈内出现一个圆点。

单选按钮的事件通告机制与其他按钮一样。当用户点击一个单选按钮时，这个按钮将产生一个动作事件。在示例中，定义了一个动作监听器用来把字体大小设置为新值：

```
ActionListener listener = new
   ActionListener()
   {
   public void actionPerformed(ActionEvent event)
   {
      // size refers to the final parameter of the addRadioButton method
      label.setFont(new Font("Serif", Font.PLAIN, size));
   }
};
```

用这个监听器与复选框中的监听器做一个对比。每个单选按钮都对应一个不同的监听器对象。每个监听器都非常清楚所要做的事情——把字体尺寸设置为一个特定值。在复选框示例中，使用的是一种不同的方法，两个复选框共享一个动作监听器。这个监听器调用一个方法来检查两个复选框的当前状态。

对于单选按钮可以使用同一个方法吗？可以试一下使用一个监听器来计算尺寸，如：

```
if (smallButton.isSelected()) size = 8;
else if (mediumButton.isSelected()) size = 12;
. . .
```

然而，更愿意使用各自独立的动作监听器，因为这样可以将尺寸值与按钮紧密地绑定在一起。

注释：如果有一组单选按钮，并知道它们之中只选择了一个。要是能够不查询组内所有的按钮就可以很快地知道哪个按钮被选择的话就好了。由于ButtonGroup对象控制着所有的按钮，所以如果这个对象能够给出被选择的按钮的引用就方便多了。事实上，ButtonGroup类中有一个getSelection方法，但是这个方法并不返回被选择的单选按钮，而是返回附加在那个按钮上的模型ButtonModel的引用。对于我们来说，ButtonModel中的方法没有什么实际的应用价值。ButtonModel接口从ItemSelectable接口继承了一个getSelectedObject方法，但是这个方法没有用，它返回null。getActionCommand方法看起来似乎可用，这是因为一个单选按钮的“动作命令”是它的文本标签，但是它的模型的动作命令是null。只有在通过setActionCommand命令明确地为所有单选按钮设定动作命令后，才能够通过调用方法buttonGroup.getSelection().getActionCommand()获得当前选择的按钮的动作命令。

例9-4是一个用于选择字体大小的完整程序，它演示了单选按钮的工作过程。

例9-4 RadioButtonTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.33 2007-06-12
7.  * @author Cay Horstmann
8.  */
9. public class RadioButtonTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                RadioButtonFrame frame = new RadioButtonFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame with a sample text label and radio buttons for selecting font sizes.
27.  */
28. class RadioButtonFrame extends JFrame
```

```
{
   public RadioButtonFrame()
   {
      setTitle("RadioButtonTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      // add the sample text label

      label = new JLabel("The quick brown fox jumps over the lazy dog.");
      label.setFont(new Font("Serif", Font.PLAIN, DEFAULT_SIZE));
      add(label, BorderLayout.CENTER);

      // add the radio buttons

      buttonPanel = new JPanel();
      group = new ButtonGroup();

      addRadioButton("Small", 8);
      addRadioButton("Medium", 12);
      addRadioButton("Large", 18);
      addRadioButton("Extra large", 36);

      add(buttonPanel, BorderLayout.SOUTH);
   }

   /**
    * Adds a radio button that sets the font size of the sample text.
    * @param name the string to appear on the button
    * @param size the font size that this button sets
    */
   public void addRadioButton(String name, final int size)
   {
      boolean selected = size == DEFAULT_SIZE;
      JRadioButton button = new JRadioButton(name, selected);
      group.add(button);
      buttonPanel.add(button);

      // this listener sets the label font size

      ActionListener listener = new ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               // size refers to the final parameter of the addRadioButton
               // method
               label.setFont(new Font("Serif", Font.PLAIN, size));
            }
         };

      button.addActionListener(listener);
   }

   public static final int DEFAULT_WIDTH = 400;
   public static final int DEFAULT_HEIGHT = 200;

   private JPanel buttonPanel;
   private ButtonGroup group;
```

```
86.    private JLabel label;
87.
88.    private static final int DEFAULT_SIZE = 12;
89. }
```

API javax.swing.JRadioButton 1.2

- JRadioButton(String label, Icon icon)
  构造一个单选按钮，初始没有被选择。
- JRadioButton(String label, boolean state)
  用给定的标签和初始状态构造一个单选按钮。

API javax.swing.ButtonGroup 1.2

- void add(AbstractButton b)
  将按钮添加到组中。
- ButtonModel getSelection()
  返回被选择的按钮的按钮模型。

API javax.swing.ButtonModel 1.2

- String getActionCommand()
  返回按钮模型的动作命令。

API javax.swing.AbstractButton 1.2

- void setActionCommand(String s)
  设置按钮及其模型的动作命令。

9.4.3 边框

如果在一个窗口中有多组单选按钮，就需要用可视化的形式指明哪些按钮属于同一组。Swing提供了一组很有用的边框（borders）来解决这个问题。可以在任何继承了JComponent的组件上应用边框。最常用的用途是在一个面板周围放置一个边框，然后用其他用户界面元素（如单选按钮）填充面板。

有几种不同的边框可供选择，但是使用它们的步骤完全一样。

1）调用BorderFactory的静态方法创建边框。下面是几种可选的风格（如图9-16所示）：

- 凹斜面
- 凸斜面
- 蚀刻
- 直线
- 不光滑

2）如果愿意的话，可以给边框添加标题，具体的实现方法是将边框传递给：

BorderFactory.createTitledBorder.

3）如果确实想把一切凸显出来，可以调用下列方法将几种边框组合起来使用：

BorderFactory.createCompoundBorder.

4）调用JComponent类中setBorder方法将结果边框添加到组件中。

例如，下面代码说明了如何把一个带有标题的蚀刻边框添加到一个面板上：

```
Border etched = BorderFactory.createEtchedBorder()
Border titled = BorderFactory.createTitledBorder(etched, "A Title");
panel.setBorder(titled);
```

运行例9-8中的程序可以看到各种边框的外观。

不同的边框有不同的用于设置边框的宽度和颜色的选项。详情请参看API注释。偏爱使用边框的人都很欣赏这一点，SoftBevelBorder类用于构造具有柔和拐角的斜面边框，LineBorder类也能够构造圆拐角。这些边框只能通过类中的某个构造器构造，而没有BorderFactory方法。

图9-16 测试边框类型

例9-5 BorderTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4. import javax.swing.border.*;
5.
6. /**
7.  * @version 1.33 2007-06-12
8.  * @author Cay Horstmann
9.  */
10. public class BorderTest
11. {
12.    public static void main(String[] args)
13.    {
14.       EventQueue.invokeLater(new Runnable()
15.          {
16.             public void run()
17.             {
18.                BorderFrame frame = new BorderFrame();
19.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.                frame.setVisible(true);
21.             }
22.          });
23.    }
24. }
25.
```

```
26. /**
27.  * A frame with radio buttons to pick a border style.
28.  */
29. class BorderFrame extends JFrame
30. {
31.    public BorderFrame()
32.    {
33.       setTitle("BorderTest");
34.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
35.
36.       demoPanel = new JPanel();
37.       buttonPanel = new JPanel();
38.       group = new ButtonGroup();
39.
40.       addRadioButton("Lowered bevel", BorderFactory.createLoweredBevelBorder());
41.       addRadioButton("Raised bevel", BorderFactory.createRaisedBevelBorder());
42.       addRadioButton("Etched", BorderFactory.createEtchedBorder());
43.       addRadioButton("Line", BorderFactory.createLineBorder(Color.BLUE));
44.       addRadioButton("Matte", BorderFactory.createMatteBorder(10, 10, 10, 10, Color.BLUE));
45.       addRadioButton("Empty", BorderFactory.createEmptyBorder());
46.
47.       Border etched = BorderFactory.createEtchedBorder();
48.       Border titled = BorderFactory.createTitledBorder(etched, "Border types");
49.       buttonPanel.setBorder(titled);
50.
51.       setLayout(new GridLayout(2, 1));
52.       add(buttonPanel);
53.       add(demoPanel);
54.    }
55.
56.    public void addRadioButton(String buttonName, final Border b)
57.    {
58.       JRadioButton button = new JRadioButton(buttonName);
59.       button.addActionListener(new ActionListener()
60.          {
61.             public void actionPerformed(ActionEvent event)
62.             {
63.                demoPanel.setBorder(b);
64.             }
65.          });
66.       group.add(button);
67.       buttonPanel.add(button);
68.    }
69.
70.    public static final int DEFAULT_WIDTH = 600;
71.    public static final int DEFAULT_HEIGHT = 200;
72.
73.    private JPanel demoPanel;
74.    private JPanel buttonPanel;
75.    private ButtonGroup group;
76. }
```

API javax.swing.BorderFactory 1.2

- static Border createLineBorder(Color color)

- static Border createLineBorder(Color color, int thickness)
  创建一个简单的直线边框。
- static MatteBorder createMatteBorder(int top, int left, int bottom, int right, Color color)
- static MatteBorder createMatteBorder(int top, int left, int bottom, int right, Icon tileIcon)
  创建一个用color颜色或一个重复（repeating）图标填充的粗的边框。
- static Border createEmptyBorder()
- static Border createEmptyBorder(int top, int left, int bottom, int right)
  创建一个空边框。
- static Border createEtchedBorder()
- static Border createEtchedBorder(Color highlight, Color shadow)
- static Border createEtchedBorder(int type)
- static Border createEtchedBorder(int type, Color highlight, Color shadow)
  创建一个具有3D效果的直线边框。
  参数：highlight, shadow　用于 3D 效果的颜色
  　　type　　EtchedBorder.RAISED和 EtchedBorder.LOWERED之一
- static Border createBevelBorder (int type)
- static Border createBevelBorder(int type, Color highlight, Color shadow)
- static Border createLoweredBevelBorder()
- static Border createRaisedBevelBorder()
  创建一个具有凹面或凸面效果的边框。
  参数：type　　BevelBorder.LOWERED和 BevelBorder.RAISED之一
  　　highlight, shadow　用于3D效果的颜色
- static TitledBorder createTitledBorder(String title)
- static TitledBorder createTitledBorder(Border border)
- static TitledBorder createTitledBorder(Border border, String title)
- static TitledBorder createTitledBorder(Border border, String title, int justification, int position)
- static TitledBorder createTitledBorder(Border border, String title, int justification, int position, Font font)
- static TitledBorder createTitledBorder(Border border, String title, int justification, int position, Font font, Color color)
  创建一个具有给定特性的带标题的边框。
  参数：title　　标题字符串
  　　border　　用标题装饰的边框
  　　justification　　TitledBorder 常量LEFT、 CENTER、RIGHT、LEADING、

	trAILING或 DEFAULT_JUSTIFICATION （left) 之一
position	TitledBorder常量ABOVE_TOP、TOP、BELOW_TOP、ABOVE_BOTTOM、 BOTTOM、 BELOW_BOTTOM或DEFAULT_POSITION (top)之一
font	标题的字体
color	标题的颜色

- `static CompoundBorder createCompoundBorder(Border outsideBorder, Border insideBorder)`
  将两个边框组合成一个新的边框。

API `javax.swing.border.SoftBevelBorder 1.2`

- `SoftBevelBorder(int type)`
- `SoftBevelBorder(int type, Color highlight, Color shadow)`
  创建一个带有柔和角的斜面边框。

参数：	type	BevelBorder.LOWERED和BevelBorder.RAISED之一
	highlight, shadow	用于3D效果的颜色

API `javax.swing.border.LineBorder 1.2`

- `public LineBorder(Color color, int thickness, boolean roundedCorners)`
  用指定的颜色和宽度创建一个直线边框。如果 roundedCorners 为true，则边框具有圆拐角。

API `javax.swing.JComponent 1.2`

- `void setBorder(Border border)`
  设置这个组件的边框。

9.4.4 组合框

如果有多个选择项，使用单选按钮就不太适宜了，其原因是占据的屏幕空间太大。这时就可以选择组合框。当用户点击这个组件时，选择列表就会下拉出来，用户可以从中选择一项（见图9-17）。

图9-17 组合框

如果下拉列表框被设置成可编辑（editable），就可以像编辑文本一样编辑当前的选项内容。鉴于这个原因，这种组件被称为组合框（combo box），它将文本域的灵活性与一组预定义的选项组合起来。JComboBox类提供了组合框的组件。

调用setEditable方法可以让组合框可编辑。注意，编辑只会影响当前项，而不会改变列表内容。

可以调用getSelectedItem方法获取当前的选项或被编辑的文本。在示例程序中，用户可以从

字体列表（Serif, SansSerif, Monospaced等）中选择一种字体，用户也可以键入其他的字体。

可以调用addItem方法增加选项。在示例程序中，只在构造器中调用了addItem方法，实际上，可以在任何地方调用它。

```
faceCombo = new JComboBox();
faceCombo.setEditable(true);
faceCombo.addItem("Serif");
faceCombo.addItem("SansSerif");
. . .
```

这个方法将字符串添加到列表的尾部。可以利用insertItemAt方法在列表的任何位置插入一个新选项：

```
faceCombo.insertItemAt("Monospaced", 0); // add at the beginning
```

可以增加任何类型的选项，组合框可以调用每个选项的toString方法显示其内容。

如果需要在运行时删除某些选项，可以使用removeItem 或者 removeItemAt 方法，使用哪个方法将取决于参数提供的是想要删除的选项内容，还是选项位置。

```
faceCombo.removeItem("Monospaced");
faceCombo.removeItemAt(0); // remove first item
```

调用removeAllItems方法将立即移除所有的选项。

提示：如果需要往组合框中添加大量的选项，addItem方法的性能就显得很差了。取而代之的是构造一个DefaultComboBoxModel，并调用addElement方法进行加载，然后再调用JComboBox中的setModel方法。

当用户从组合框中选择一个选项时，组合框就将产生一个动作事件。为了判断哪个选项被选择，可以通过事件参数调用getSource方法来得到发送事件的组合框引用，接着调用getSelectedItem方法获取当前选择的选项。需要把这个方法的返回值转化为相应的类型，通常是String型。

```
public void actionPerformed(ActionEvent event)
{
   label.setFont(new Font(
      (String) faceCombo.getSelectedItem(),
      Font.PLAIN,
      DEFAULT_SIZE));
}
```

例9-6给出了完整的代码。

注释：如果希望持久地显示列表，而不是下拉列表，就应该使用JList组件。在卷II的第6章中将介绍JList。

例9-6 ComboBoxTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
```

```
/**
 * @version 1.33 2007-06-12
 * @author Cay Horstmann
 */
public class ComboBoxTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(new Runnable()
         {
            public void run()
            {

               ComboBoxFrame frame = new ComboBoxFrame();
               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
               frame.setVisible(true);
            }
         });
   }
}

/**
 * A frame with a sample text label and a combo box for selecting font faces.
 */
class ComboBoxFrame extends JFrame
{
   public ComboBoxFrame()
   {
      setTitle("ComboBoxTest");
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

      // add the sample text label

      label = new JLabel("The quick brown fox jumps over the lazy dog.");
      label.setFont(new Font("Serif", Font.PLAIN, DEFAULT_SIZE));
      add(label, BorderLayout.CENTER);

      // make a combo box and add face names

      faceCombo = new JComboBox();
      faceCombo.setEditable(true);
      faceCombo.addItem("Serif");
      faceCombo.addItem("SansSerif");
      faceCombo.addItem("Monospaced");
      faceCombo.addItem("Dialog");
      faceCombo.addItem("DialogInput");

      // the combo box listener changes the label font to the selected face name

      faceCombo.addActionListener(new ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               label.setFont(new Font((String) faceCombo.getSelectedItem(), Font.PLAIN,
                     DEFAULT_SIZE));
            }
         });
```

```
62.
63.       // add combo box to a panel at the frame's southern border
64.
65.       JPanel comboPanel = new JPanel();
66.       comboPanel.add(faceCombo);
67.       add(comboPanel, BorderLayout.SOUTH);
68.    }
69.
70.    public static final int DEFAULT_WIDTH = 300;
71.    public static final int DEFAULT_HEIGHT = 200;
72.
73.    private JComboBox faceCombo;
74.    private JLabel label;
75.    private static final int DEFAULT_SIZE = 12;
76. }
```

API javax.swing.JComboBox 1.2

- boolean isEditable()
- void setEditable(boolean b)
  获取或设置组合框的可编辑特性。
- void addItem(Object item)
  把一个选项添加到选项列表中。
- void insertItemAt(Object item, int index)
  将一个选项添加到选项列表的指定位置。
- void removeItem(Object item)
  从选项列表中删除一个选项。
- void removeItemAt(int index)
  删除指定位置的选项。
- void removeAllItems()
  从选项列表中删除所有选项。
- Object getSelectedItem()
  返回当前选择的选项。

9.4.5 滑块

组合框可以让用户从一组离散值中进行选择。滑块允许进行连续值得选择，例如，从1～100之间选择任意数值。

通常，可以使用下列方式构造滑块：

```
JSlider slider = new JSlider(min, max, initialValue);
```

如果省略最小值、最大值和初始值，其默认值分别为0、100和50。

如果需要垂直滑块，可以按照下列方式调用构造器：

```
JSlider slider = new JSlider(SwingConstants.VERTICAL, min, max, initialValue);
```

这个构造器构造了一个无格式的滑块，如同图9-18最上面的滑块所示。下面看一下如何为滑块添加装饰。

当用户滑动滑块时，滑块的值就会在最小值和最大值之间变化。当值发生变化时，ChangeEvent就会发送给所有变化的监听器。为了得到这些改变的通知，需要调用addChangeListener方法并且安装一个实现了ChangeListener接口的对象。这个接口只有一个方法StateChanged。在这个方法中，可以获取滑块的当前值：

```
public void stateChanged(ChangeEvent event)
{
   JSlider slider = (JSlider) event.getSource();
   int value = slider.getValue();
   . . .
}
```

可以通过显示标尺（tricks）对滑块进行修饰。例如，在示例程序中，第二个滑块使用了下面的设置：

```
slider.setMajorTickSpacing(20);
slider.setMinorTickSpacing(5);
```

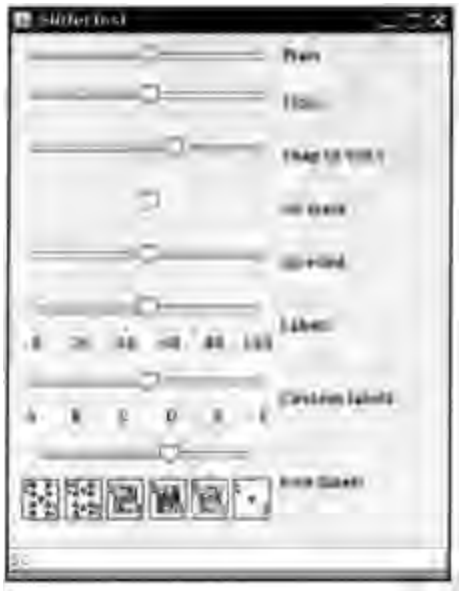


图9-18 滑块

上述滑块在每20个单元的位置显示一个大标尺标记，每5个单元的职位显示一个小标尺标记。所谓单元是指滑块值，而不是像素。

这些代码只设置了标尺标记，要想将它们显示出来，还需要调用：

```
slider.setPaintTicks(true);
```

大标尺和小标尺是相互独立的。例如，可以每20个单元设置一个大标尺，同时每7个单元设置一个小标尺，但是这样设置，滑块看起来会显得非常凌乱。

可以强制滑块对齐标尺。这样一来，只要用户完成拖放滑块的操作，滑块就会立即自动地移到最接近的标尺处。激活这种操作方式需要调用：

```
slider.setSnapToTicks(true);
```

注释：“对齐标尺”的行为与想像的工作过程并不太一样。在滑块真正对齐之前，改变监听器报告的滑块值并不是对应的标尺值。如果点击了滑块附近，滑块将会向点击的方向移动一小段距离，“对齐标尺”的滑块并不移动到下一个标尺处。

可以调用下列方法为大标尺添加标尺标签(tick mark labels)：

```
slider.setPaintLabels(true);
```

例如，对于一个范围为0到100的滑块，如果大标尺的间距是20，每个大标尺的标签就应该分别是0、20、40、60、80和100。

还可以提供其他形式的标尺标记，如字符串或者图标（见图9-18）。这样做有些烦琐。首先需要填充一个键为Integer类型且值为Component类型的散列表（在JDK5.0中自打包让这个过程变得十分容易）。然后再调用setLabelTable方法，组件就会放置在标尺标记处。通常组件使用的是JLabel对象。下面代码说明了如何将标尺标签设置为A、B、C、D、E和F。

```
Hashtable<Integer, Component> labelTable = new Hashtable<Integer, Component>();
labelTable.put(0, new JLabel("A"));
labelTable.put(20, new JLabel("B"));
. . .
labelTable.put(100, new JLabel("F"));
slider.setLabelTable(labelTable);
```

关于散列表的详细介绍，参考卷II的第2章。

例9-7显示了如何创建用图标作为标尺标签的滑块。

提示：如果标尺的标记或者标签不显示，请检查一下是否调用了setPaintTicks(true)和setPaintLabels(true)。

在图9-18中，第4个滑块没有轨迹。要想隐藏滑块移动的轨迹，可以调用：

```
slider.setPaintTrack(false);
```

图9-18中第5个滑块是逆向的，调用下列方法可以实现这个效果：

```
slider.setInverted(true);
```

示例程序演示了所有不同视觉效果的滑块。每个滑块都安装了一个改变事件监听器，它负责把当前的滑块值显示到框架底部的文本域中。

例9-7 SliderTest.java

```
1. import java.awt.*;
2. import java.util.*;
3. import javax.swing.*;
4. import javax.swing.event.*;
5.
6. /**
7.  * @version 1.13 2007-06-12
8.  * @author Cay Horstmann
9.  */
10. public class SliderTest
11. {
12.    public static void main(String[] args)
13.    {
14.       EventQueue.invokeLater(new Runnable()
15.          {
16.             public void run()
17.             {
18.                SliderTestFrame frame = new SliderTestFrame();
19.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.                frame.setVisible(true);
21.             }
22.          });
23.    }
24. }
25.
26. /**
27.  * A frame with many sliders and a text field to show slider values.
28.  */
29. class SliderTestFrame extends JFrame
30. {
31.    public SliderTestFrame()
```

```
32.    {
33.       setTitle("SliderTest");
34.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
35.
36.       sliderPanel = new JPanel();
37.       sliderPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
38.
39.       // common listener for all sliders
40.       listener = new ChangeListener()
41.          {
42.             public void stateChanged(ChangeEvent event)
43.             {
44.                // update text field when the slider value changes
45.                JSlider source = (JSlider) event.getSource();
46.                textField.setText("" + source.getValue());
47.             }
48.          };
49.
50.       // add a plain slider
51.
52.       JSlider slider = new JSlider();
53.       addSlider(slider, "Plain");
54.
55.       // add a slider with major and minor ticks
56.
57.       slider = new JSlider();
58.       slider.setPaintTicks(true);
59.       slider.setMajorTickSpacing(20);
60.       slider.setMinorTickSpacing(5);
61.       addSlider(slider, "Ticks");
62.
63.       // add a slider that snaps to ticks
64.
65.       slider = new JSlider();
66.       slider.setPaintTicks(true);
67.       slider.setSnapToTicks(true);
68.       slider.setMajorTickSpacing(20);
69.       slider.setMinorTickSpacing(5);
70.       addSlider(slider, "Snap to ticks");
71.
72.       // add a slider with no track
73.
74.       slider = new JSlider();
75.       slider.setPaintTicks(true);
76.       slider.setMajorTickSpacing(20);
77.       slider.setMinorTickSpacing(5);
78.       slider.setPaintTrack(false);
79.       addSlider(slider, "No track");
80.
81.       // add an inverted slider
82.
83.       slider = new JSlider();
84.       slider.setPaintTicks(true);
85.       slider.setMajorTickSpacing(20);
86.       slider.setMinorTickSpacing(5);
87.       slider.setInverted(true);
88.       addSlider(slider, "Inverted");
```

```
89.
90.       // add a slider with numeric labels
91.
92.       slider = new JSlider();
93.       slider.setPaintTicks(true);
94.       slider.setPaintLabels(true);
95.       slider.setMajorTickSpacing(20);
96.       slider.setMinorTickSpacing(5);
97.       addSlider(slider, "Labels");
98.
99.       // add a slider with alphabetic labels
100.
101.      slider = new JSlider();
102.      slider.setPaintLabels(true);
103.      slider.setPaintTicks(true);
104.      slider.setMajorTickSpacing(20);
105.      slider.setMinorTickSpacing(5);
106.
107.      Dictionary<Integer, Component> labelTable = new Hashtable<Integer, Component>();
108.      labelTable.put(0, new JLabel("A"));
109.      labelTable.put(20, new JLabel("B"));
110.      labelTable.put(40, new JLabel("C"));
111.      labelTable.put(60, new JLabel("D"));
112.      labelTable.put(80, new JLabel("E"));
113.      labelTable.put(100, new JLabel("F"));
114.
115.      slider.setLabelTable(labelTable);
116.      addSlider(slider, "Custom labels");
117.
118.      // add a slider with icon labels
119.
120.      slider = new JSlider();
121.      slider.setPaintTicks(true);
122.      slider.setPaintLabels(true);
123.      slider.setSnapToTicks(true);
124.      slider.setMajorTickSpacing(20);
125.      slider.setMinorTickSpacing(20);
126.
127.      labelTable = new Hashtable<Integer, Component>();
128.
129.      // add card images
130.
131.      labelTable.put(0, new JLabel(new ImageIcon("nine.gif")));
132.      labelTable.put(20, new JLabel(new ImageIcon("ten.gif")));
133.      labelTable.put(40, new JLabel(new ImageIcon("jack.gif")));
134.      labelTable.put(60, new JLabel(new ImageIcon("queen.gif")));
135.      labelTable.put(80, new JLabel(new ImageIcon("king.gif")));
136.      labelTable.put(100, new JLabel(new ImageIcon("ace.gif")));
137.
138.      slider.setLabelTable(labelTable);
139.      addSlider(slider, "Icon labels");
140.
141.      // add the text field that displays the slider value
142.
143.      textField = new JTextField();
144.      add(sliderPanel, BorderLayout.CENTER);
145.      add(textField, BorderLayout.SOUTH);
```

```
146.    }
147.
148.    /**
149.     * Adds a slider to the slider panel and hooks up the listener
150.     * @param s the slider
151.     * @param description the slider description
152.     */
153.    public void addSlider(JSlider s, String description)
154.    {
155.       s.addChangeListener(listener);
156.       JPanel panel = new JPanel();
157.       panel.add(s);
158.       panel.add(new JLabel(description));
159.       sliderPanel.add(panel);
160.    }
161.
162.    public static final int DEFAULT_WIDTH = 350;
163.    public static final int DEFAULT_HEIGHT = 450;
164.
165.    private JPanel sliderPanel;
166.    private JTextField textField;
167.    private ChangeListener listener;
168. }
```

API `javax.swing.JSlider` 1.2

- `JSlider()`
- `JSlider(int direction)`
- `JSlider(int min, int max)`
- `JSlider(int min, int max, int initialValue)`
- `JSlider(int direction, int min, int max, int initialValue)`

用给定的方向、最大值、最小值和初始化值构造一个水平滑块。

参数：direction　SwingConstants.HORIZONTAL 或SwingConstants.VERTICAL之一。默认为水平

min, max　滑块的最大值、最小值。默认值为0到100

initialValue　滑块的初始化值。默认值为50

- `void setPaintTicks(boolean b)`

如果b为true，显示标尺。

- `void setMajorTickSpacing(int units)`
- `void setMinorTickSpacing(int units)`

用给定的滑块单元的倍数设置大标尺和小标尺。

- `void setPaintLabels(boolean b)`

如果b是true，显示标尺标签。

- `void setLabelTable(Dictionary table)`

设置用于作为标尺标签的组件。表中的每一个键/值对都采用new Integer(value)/component

的格式。

- `void setSnapToTicks(boolean b)`
  如果b是true，每一次调整后滑块都要对齐到最接近的标尺处。
- `void setPaintTrack(boolean b)`
  如果b是true，显示滑块滑动的轨迹。

9.5 菜单

前面介绍了几种最常用的可以放到窗口内的组件，如：各种按钮、文本域以及组合框等。Swing还提供了一些其他种类的用户界面元素，下拉式菜单就是GUI应用程序中最常见的一种。

图9-19 带有子菜单的菜单

位于窗口顶部的菜单栏（menu bar）包括了下拉菜单的名字。点击一个名字就可以打开包含菜单项（menu items）和子菜单（submenus）的菜单。当用户点击菜单项时，所有的菜单都会被关闭并且将一条消息发送给程序。图9-19显示了一个带子菜单的典型菜单。

9.5.1 菜单创建

创建菜单是一件非常容易的事情。首先要创建一个菜单栏：

```
JMenuBar menuBar = new JMenuBar();
```

菜单栏是一个可以添加到任何位置的组件。通常放置在框架的顶部。可以调用setJMenuBar方法将菜单栏添加到框架上：

```
frame.setJMenuBar(menuBar);
```

需要为每个菜单建立一个菜单对象：

```
JMenu editMenu = new JMenu("Edit");
```

然后将顶层菜单添加到菜单栏中：

```
menuBar.add(editMenu);
```

向菜单对象中添加菜单项、分隔符和子菜单：

```
JMenuItem pasteItem = new JMenuItem("Paste");
editMenu.add(pasteItem);
editMenu.addSeparator();
JMenu optionsMenu = . . .; // a submenu
editMenu.add(optionsMenu);
```

可以看到分隔符位于Paste和Read-only菜单项之间。

当用户选择菜单时，将触发一个动作事件。这里需要为每个菜单项安装一个动作监听器。

```
ActionListener listener = . . .;
pasteItem.addActionListener(listener);
```

可以使用JMenu.add(String s)方法将菜单项插入到菜单的尾部，例如：

```
editMenu.add("Paste");
```

Add方法返回创建的子菜单项。可以采用下列方式获取它，并添加监听器：

```
JMenuItem pasteItem = editMenu.add("Paste");
pasteItem.addActionListener(listener);
```

在通常情况下，菜单项触发的命令也可以通过其他用户界面元素（如工具栏上的按钮）激活。在第8章中，已经看到了如何通过Action对象来指定命令。通常，采用扩展抽象类AbstractAction来定义一个实现Action接口的类。这里需要在AbstractAction对象的构造器中指定菜单项标签并且覆盖actionPerformed方法来获得菜单动作处理器。例如：

```
Action exitAction = new AbstractAction("Exit") // menu item text goes here
   {
      public void actionPerformed(ActionEvent event)
      {
         // action code goes here
         System.exit(0);
      }
   };
```

然后将动作添加到菜单中：

```
JMenuItem exitItem = fileMenu.add(exitAction);
```

这个命令利用动作名将一个菜单项添加到菜单中。这个动作对象将作为它的监听器。上面这条语句是下面两条语句的缩写形式：

```
JMenuItem exitItem = new JMenuItem(exitAction);
fileMenu.add(exitItem);
```

注释：在Windows和Macintosh程序中，通常菜单定义在外部资源文件中，并利用资源标识符将其绑定到应用程序。在Java中，菜单通常在程序内部创建，这是因为Java处理外部资源的机制比Windows和Mac操作系统的限制更多。

`javax.swing.JMenu 1.2`

- `JMenu(String label)`
  用给定标签构造一个菜单。
- `JMenuItem add(JMenuItem item)`
  添加一个菜单项（或一个菜单）。
- `JMenuItem add(String label)`
  用给定标签将一个菜单项添加到菜单中，并返回这个菜单项。
- `JMenuItem add(Action a)`
  用给定动作将一个菜单项添加到菜单中，并返回这个菜单项。
- `void addSeparator()`
  将一个分隔符行（ separator line ）添加到菜单中。
- `JMenuItem insert(JMenuItem menu, int index)`

将一个新菜单项（或子菜单）添加到菜单的指定位置。

- `JMenuItem insert(Action a, int index)`
  用给定动作在菜单的指定位置添加一个新菜单项。
- `void insertSeparator(int index)`
  将一个分隔符添加到菜单中。
  参数：index　　添加分隔符的位置
- `void remove(int index)`
- `void remove(JMenuItem item)`
  从菜单中删除指定的菜单项。

API `javax.swing.JMenuItem 1.2`

- `JMenuItem(String label)`
  用给定标签构造一个菜单项。
- `JMenuItem(Action a) 1.3`
  为给定动作构造一个菜单项。

API `javax.swing.AbstractButton 1.2`

- `void setAction(Action a) 1.3`
  为这个按钮或菜单项设置动作。

API `javax.swing.JFrame 1.2`

- `void setJMenuBar(JMenuBar menubar)`
  为这个框架设置菜单栏。

9.5.2 菜单项中的图标

菜单项与按钮很相似。实际上，JMenuItem类扩展了AbstractButton类。与按钮一样，菜单可以包含文本标签、图标，也可以两者都包含。既可以利用JMenuItem(String, Icon)或者JMenuItem(Icon) 构造器为菜单指定一个图标，也可以利用JMenuItem类中的setIcon方法（继承自AbstractButton类）指定一个图标。例如：

```
JMenuItem cutItem = new JMenuItem("Cut", new ImageIcon("cut.gif"));
```

图9-19展示了具有图标的菜单项。正如所看到的，在默认情况下，菜单项的文本被放置在图标的右侧。如果喜欢将文本放置在左侧，可以调用JMenuItem类中的setHorizontalTextPosition方法（继承自AbstractButton类）设置。例如：

```
cutItem.setHorizontalTextPosition(SwingConstants.LEFT);
```

这个调用把菜单项文本移动到图标的左侧。

也可以将一个图标添加到一个动作上：

```
cutAction.putValue(Action.SMALL_ICON, new ImageIcon("cut.gif"));
```

当使用动作构造菜单项时，Action.NAME值将会作为菜单项的文本，而Action.SMALL_ICON将会作为图标。

另外，可以利用AbstractAction构造器设置图标：

```
cutAction = new
   AbstractAction("Cut", new ImageIcon("cut.gif"))
   {
      public void actionPerformed(ActionEvent event)
      {
         // action code goes here
      }
   };
```

API javax.swing.JMenuItem 1.2

- JMenuItem(String label, Icon icon)
  用给定的标签和图标构造一个菜单项。

API javax.swing.AbstractButton 1.2

- void setHorizontalTextPosition(int pos)
  设置文本对应图标的水平位置。
  参数：pos SwingConstants.RIGHT（文本在图标的右侧）或 SwingConstants.LEFT

API javax.swing.AbstractAction 1.2

- AbstractAction(String name, Icon smallIcon)
  用给定的名字和图标构造一个抽象的动作。

9.5.3 复选框和单选按钮菜单项

复选框和单选按钮菜单项在文本旁边显示了一个复选框或一个单选按钮（参见图9-19）。当用户选择一个菜单项时，菜单项就会自动地在选择和未选择间进行切换。

除了按钮装饰外，同其他菜单项的处理一样。例如，下面是创建复选框菜单项的代码：

```
JCheckBoxMenuItem readonlyItem = new JCheckBoxMenuItem("Read-only");
optionsMenu.add(readonlyItem);
```

单选按钮菜单项与普通单选按钮的工作方式一样，必须将它们加入到按钮组中。当按钮组中的一个按钮被选中时，其他按钮都自动地变为未选择项。

```
ButtonGroup group = new ButtonGroup();
JRadioButtonMenuItem insertItem = new JRadioButtonMenuItem("Insert");
insertItem.setSelected(true);
JRadioButtonMenuItem overtypeItem = new JRadioButtonMenuItem("Overtype");
group.add(insertItem);
group.add(overtypeItem);
optionsMenu.add(insertItem);
optionsMenu.add(overtypeItem);
```

使用这些菜单项，不需要立刻得到用户选择菜单项的通知。而是使用isSelected方法来测试

菜单项的当前状态（当然，这意味着应该保留一个实例域保存这个菜单项的引用）。使用setSelect方法设置状态。

API javax.swing.JCheckBoxMenuItem 1.2

- JCheckBoxMenuItem(String label)
  用给定的标签构造一个复选框菜单项。
- JCheckBoxMenuItem(String label, boolean state)
  用给定的标签和给定的初始状态（true为选定）构造一个复选框菜单。

API javax.swing.JRadioButtonMenuItem 1.2

- JRadioButtonMenuItem(String label)
  用给定的标签构造一个单选按钮菜单项。
- JRadioButtonMenuItem(String label, boolean state)
  用给定的标签和给定的初始状态（true为选定）构造一个单选按钮菜单项。

API javax.swing.AbstractButton 1.2

- boolean isSelected()
- void setSelected(boolean state)
  获取或设置这个菜单项的选择状态（true为选定）。

9.5.4 弹出菜单

弹出菜单（pop-up menu）是不固定在菜单栏中随处浮动的菜单（参见图9-20）

图9-20 弹出菜单

创建一个弹出菜单与创建一个常规菜单的方法类似，但是弹出菜单没有标题。

```
JPopupMenu popup = new JPopupMenu();
```

然后用常规的方法添加菜单项：

```
JMenuItem item = new JMenuItem("Cut");
item.addActionListener(listener);
popup.add(item);
```

弹出菜单并不像常规菜单栏那样总是显示在框架的顶部，必须调用show方法菜单才能显示出来。调用时需要给出父组件以及相对父组件坐标的显示位置。例如：

```
popup.show(panel, x, y);
```

通常，当用户点击某个鼠标键时弹出菜单。这就是所谓的弹出式触发器（pop-up trigger）。在Windows或者Linux中，弹出式触发器是鼠标右键。要想在用户点击某一个组件时弹出菜单，需要按照下列方式调用方法：

```
component.setComponentPopupMenu(popup);
```

偶尔会遇到在一个含有弹出菜单的组件中放置一个组件的情况。这个子组件可以调用下列方法继承父组件的弹出菜单。调用：

```
child.setInheritsPopupMenu(true);
```

这些方法是SE5.0中新增加的，这样可以让程序员不必关心系统对弹出菜单的依赖。在SE5.0之前，必须安装鼠标监听器，并将下列代码添加到mousePressed和mouseReleased监听器中：

```
if (popup.isPopupTrigger(event))
   popup.show(event.getComponent(), event.getX(), event.getY());
```

有些系统在按钮压下时触发弹出菜单，有些系统则按钮抬起时触发。

API `javax.swing.JPopupMenu 1.2`

- `void show(Component c, int x, int y)`
  显示一个弹出菜单。
  参数：c　　显示弹出菜单的组件
  　　x, y　　弹出菜单左上角的坐标（c的坐标空间内）
- `boolean isPopupTrigger(MouseEvent event) 1.3`
  如果鼠标事件是弹出菜单触发器，则返回 true。

API `java.awt.event.MouseEvent 1.1`

- `boolean isPopupTrigger()`
  如果鼠标事件是弹出菜单触发器，则返回 true 。

API `javax.swing.JComponent 1.2`

- `JPopupMenu getComponentPopupMenu() 5.0`
- `void setComponentPopupMenu(JPopupMenu popup) 5.0`
  获取或设置用于这个组件的弹出菜单。
- `boolean getInheritsPopupMenu() 5.0`
- `void setInheritsPopupMenu(boolean b) 5.0`
  获取或设置 inheritsPopupMenu特性。如果这个特性被设置或这个组件的弹出菜单为null，则应用上一级弹出菜单。

9.5.5 快捷键和加速器

对于有经验的用户来说，通过快捷键来选择菜单项会感觉更加便捷。可以通过在菜单项的构造器中指定一个快捷字母来为菜单项设置快捷键：

```
JMenuItem aboutItem = new JMenuItem("About", 'A');
```

快捷键会自动地显示在菜单项中，并带有一条下划线（如图9-21）。例如，在上面的例子中，菜单项中的标签为“ About ”，

图9-21 键盘快捷键

字母A带有一个下划线。当显示菜单时，用户只需要按下“A”键就可以这个选择菜单项（如果快捷字母没有出现在菜单项标签字符串中，同样可以按下快捷键选择菜单项，只是快捷键没有显示出来。很自然，这种不可见的快捷键没有提示效果）。

有时候不希望在菜单项的第一个快捷键字母下面加下划线。例如，如果在菜单项“Save As”中使用快捷键“A”，则在第二个“A”（Save As）下面加下划线更为合理。在Java SE1.4中，可以调用setDisplayedMnemonicIndex方法指定希望加下划线的字符。

如果有一个Action对象，就可以把快捷键作为Action. MNEMONIC_KEY的键值添加到对象中。如：

```
cutAction.putValue(Action.MNEMONIC_KEY, new Integer('A'));
```

只能在菜单项的构造器中设定快捷键字母，而不是在菜单构造器中。如果想为菜单设置快捷键，需要调用setMnemonic方法：

```
JMenu helpMenu = new JMenu("Help");
helpMenu.setMnemonic('H');
```

可以同时按下ALT键和菜单的快捷键来实现在菜单栏中选择一个顶层菜单的操作。例如：按下ALT+H可以从菜单中选择Help菜单项。

可以使用快捷键从当前打开的菜单中选择一个子菜单或者菜单项。而加速器是在不打开菜单的情况下选择菜单项的快捷键。例如：很多程序把加速器CTRL+O和CTRL+S关联到File菜单中的Open和Save菜单项。可以使用setAccelerator将加速器键关联到一个菜单项上。这个方法使用KeyStroke类型的对象作为参数。例如：下面的调用将加速器CTRL+O关联到OpenItem菜单项。

```
openItem.setAccelerator(KeyStroke.getKeyStroke("ctrl O"));
```

当用户按下加速器组合键时，就会自动地选择相应的菜单项，同时激活一个动作事件，这与手工地选择这个菜单项一样。

加速器只能关联到菜单项上，不能关联到菜单上。加速器键并不实际打开菜单。它将直接地激活菜单关联的动作事件。

从概念上讲，把加速器添加到菜单项与把加速器添加到Swing组件上所使用的技术十分类似（在第8章中讨论了这个技术）。但是，当加速器添加到菜单项时，对应的组合键就会自动地显示在相应的菜单上（如图9-22）。

图9-22 加速键

注释：在Windows下，ALT+F4用于关闭窗口。但这不是Java程序设定的加速键。这是操作系统定义的快捷键。这个组合键总会触发活动窗口的WindowClosing事件，而不管菜单上是否有Close菜单项。

javax.swing.JMenuItem 1.2

- `JMenuItem(String label, int mnemonic)`

  用给定的标签和快捷键字符构造一个菜单项

  参数：label 菜单项的标签

  mnemonic 菜单项的快捷键字符，在标签中这个字符下面会有一个下划线。

- `void setAccelerator(KeyStroke k)`
  将k设置为这个菜单项的加速器。加速器显示在标签的旁边。

API `javax.swing.AbstractButton 1.2`

- `void setMnemonic(int mnemonic)`
  设置按钮的快捷字符。该字符会在标签中以下划线的形式显示。
  参数：mnemonic 按钮的快捷字符。
- `void setDisplayedMnemonicIndex(int index) 1.4`
  将按钮文本中的index字符设定为带下划线。如果不希望第一个出现的快捷键字符带下划线，就可以使用这个方法。
  参数：index 在按钮文本中设定为带下划线的字符索引。

9.5.6 启用和禁用菜单项

在有些时候，某个特定的菜单项可能只能够在某种特定的环境下才可用。例如，当文档以只读方式打开时，Save菜单项就没有意义。当然，可以使用JMenu.remove方法将这个菜单项从菜单中删掉，但用户会对菜单内容的不断变化感到奇怪。然而，可以将这个菜单项设为禁用状态，以便屏蔽掉这些暂时不适用的命令。被禁用的菜单项被显示为灰色，不能被选择它（如图9-23）。

图9-23 禁用菜单项

启用或禁用菜单项需要调用setEnabled方法：

```
saveItem.setEnabled(false);
```

启用和禁用菜单项有两种策略。每次环境发生变化就对相关的菜单项或动作调用setEnabled。例如：只要当文档以只读方式打开，就禁用Save和Save As菜单项。另一种方法是在显示菜单之前禁用这些菜单项。这里必须为“menu selected”事件注册监听器。javax.swing.event包定义了MenuListener接口，它包含三个方法：

```
void menuSelected(MenuEvent event)
void menuDeselected(MenuEvent event)
void menuCanceled(MenuEvent event)
```

由于在菜单显示之前调用menuSelected方法，所以可以在这个方法中禁用或启用菜单项。下面代码显示了只读复选框菜单项被选择以后，如何禁用Save和Save As动作。

```
public void menuSelected(MenuEvent event)
{
   saveAction.setEnabled(!readonlyItem.isSelected());
   saveAsAction.setEnabled(!readonlyItem.isSelected());
}
```

警告：在显示菜单之前禁用菜单项是一种明智的选择，但这种方式不适用于带有加速键的菜单项。这是因为在敲击加速键时并没有打开菜单，因此动作没有被禁用，致使加速键还会触发这个行为。

API javax.swing.JMenuItem 1.2

- void setEnabled(boolean b)
  启用或禁用菜单项。

API javax.swing.event.MenuListener 1.20

- void menuSelected(MenuEvent e)
  在菜单被选择但尚未打开之前调用。
- void menuDeselected(MenuEvent e)
  在菜单被取消选择并且已经关闭之后被调用。
- void menuCanceled(MenuEvent e)
  当菜单被取消时被调用。例如，用户点击菜单以外的区域。

例9-8是创建一组菜单的示例程序。这个程序演示了本节介绍的所有特性，包括：嵌套菜单、禁用菜单项、复选框和单选按钮菜单项、弹出菜单以及快捷键和加速器。

例9-8 MenuTest.java

```
1. import java.awt.EventQueue;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.23 2007-05-30
7.  * @author Cay Horstmann
8.  */
9. public class MenuTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                MenuFrame frame = new MenuFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame with a sample menu bar.
27.  */
28. class MenuFrame extends JFrame
29. {
30.    public MenuFrame()
31.    {
32.       setTitle("MenuTest");
33.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
```

```
34.
35.       JMenu fileMenu = new JMenu("File");
36.       fileMenu.add(new TestAction("New"));
37.
38.       // demonstrate accelerators
39.
40.       JMenuItem openItem = fileMenu.add(new TestAction("Open"));
41.       openItem.setAccelerator(KeyStroke.getKeyStroke("ctrl O"));
42.
43.       fileMenu.addSeparator();
44.
45.       saveAction = new TestAction("Save");
46.       JMenuItem saveItem = fileMenu.add(saveAction);
47.       saveItem.setAccelerator(KeyStroke.getKeyStroke("ctrl S"));
48.
49.       saveAsAction = new TestAction("Save As");
50.       fileMenu.add(saveAsAction);
51.       fileMenu.addSeparator();
52.
53.       fileMenu.add(new AbstractAction("Exit")
54.          {
55.             public void actionPerformed(ActionEvent event)
56.             {
57.                System.exit(0);
58.             }
59.          });
60.
61.       // demonstrate check box and radio button menus
62.
63.       readonlyItem = new JCheckBoxMenuItem("Read-only");
64.       readonlyItem.addActionListener(new ActionListener()
65.          {
66.             public void actionPerformed(ActionEvent event)
67.             {
68.                boolean saveOk = !readonlyItem.isSelected();
69.                saveAction.setEnabled(saveOk);
70.                saveAsAction.setEnabled(saveOk);
71.             }
72.          });
73.
74.       ButtonGroup group = new ButtonGroup();
75.
76.       JRadioButtonMenuItem insertItem = new JRadioButtonMenuItem("Insert");
77.       insertItem.setSelected(true);
78.       JRadioButtonMenuItem overtypeItem = new JRadioButtonMenuItem("Overtype");
79.
80.       group.add(insertItem);
81.       group.add(overtypeItem);
82.
83.       // demonstrate icons
84.
85.       Action cutAction = new TestAction("Cut");
86.       cutAction.putValue(Action.SMALL_ICON, new ImageIcon("cut.gif"));
87.       Action copyAction = new TestAction("Copy");
88.       copyAction.putValue(Action.SMALL_ICON, new ImageIcon("copy.gif"));
89.       Action pasteAction = new TestAction("Paste");
90.       pasteAction.putValue(Action.SMALL_ICON, new ImageIcon("paste.gif"));
```

```
91.
92.       JMenu editMenu = new JMenu("Edit");
93.       editMenu.add(cutAction);
94.       editMenu.add(copyAction);
95.       editMenu.add(pasteAction);
96.
97.       // demonstrate nested menus
98.
99.       JMenu optionMenu = new JMenu("Options");
100.
101.      optionMenu.add(readonlyItem);
102.      optionMenu.addSeparator();
103.      optionMenu.add(insertItem);
104.      optionMenu.add(overtypeItem);
105.
106.      editMenu.addSeparator();
107.      editMenu.add(optionMenu);
108.
109.      // demonstrate mnemonics
110.
111.      JMenu helpMenu = new JMenu("Help");
112.      helpMenu.setMnemonic('H');
113.
114.      JMenuItem indexItem = new JMenuItem("Index");
115.      indexItem.setMnemonic('I');
116.      helpMenu.add(indexItem);
117.
118.      // you can also add the mnemonic key to an action
119.      Action aboutAction = new TestAction("About");
120.      aboutAction.putValue(Action.MNEMONIC_KEY, new Integer('A'));
121.      helpMenu.add(aboutAction);
122.
123.      // add all top-level menus to menu bar
124.
125.      JMenuBar menuBar = new JMenuBar();
126.      setJMenuBar(menuBar);
127.
128.      menuBar.add(fileMenu);
129.      menuBar.add(editMenu);
130.      menuBar.add(helpMenu);
131.
132.      // demonstrate pop-ups
133.
134.      popup = new JPopupMenu();
135.      popup.add(cutAction);
136.      popup.add(copyAction);
137.      popup.add(pasteAction);
138.
139.      JPanel panel = new JPanel();
140.      panel.setComponentPopupMenu(popup);
141.      add(panel);
142.
143.      // the following line is a workaround for bug 4966109
144.      panel.addMouseListener(new MouseAdapter()
145.         {
146.         });
147.   }
```

```
148.
149.    public static final int DEFAULT_WIDTH = 300;
150.    public static final int DEFAULT_HEIGHT = 200;
151.
152.    private Action saveAction;
153.    private Action saveAsAction;
154.    private JCheckBoxMenuItem readonlyItem;
155.    private JPopupMenu popup;
156. }
157.
158. /**
159.  * A sample action that prints the action name to System.out
160.  */
161. class TestAction extends AbstractAction
162. {
163.    public TestAction(String name)
164.    {
165.       super(name);
166.    }
167.
168.    public void actionPerformed(ActionEvent event)
169.    {
170.       System.out.println(getValue(Action.NAME) + " selected.");
171.    }
172. }
```

9.5.7 工具栏

图9-24 工具栏

工具栏是在程序中提供的快速访问常用命令的按钮栏，如图9-24所示。

工具栏的特殊之处在于可以将它随处移动。可以将它拖拽到框架的四个边框上，如图9-25所示。释放鼠标按钮后，工具栏将会停靠在新的位置上，如图9-26所示。

图9-25 拖拽工具栏

图9-26 将工具栏拖拽到另一边框

注释：工具栏只有位于采用边框布局或者任何支持North、East、South和West约束布局管理器的容器内才能够被拖拽。

工具栏可以完全脱离框架。这样的工具栏将包含在自己的框架中，如图9-27所示。当关闭包含工具栏的框架时，它会回到原始的框架中。

编写创建工具栏的代码非常容易，并且可以将组件添加到工具栏中：

```
JToolBar bar = new JToolBar();
bar.add(blueButton);
```

图9-27 脱离的工具栏

JToolBar类还有一个用来添加Action对象的方法，可以用Action对象填充工具栏：

```
bar.add(blueAction);
```

这个动作的小图标将会出现在工具栏中。

可以用分隔符将按钮分组：

```
bar.addSeparator();
```

例如，图9-24中的工具栏有一个分隔符，它位于第三个按钮和第四个按钮之间。

然后，将工具栏添加到框架中：

```
add(bar, BorderLayout.NORTH);
```

当工具栏没有停靠时，可以指定工具栏的标题：

```
bar = new JToolBar(titleString);
```

在默认情况下，工具栏最初为水平的。如果想要将工具栏垂直放置，可以使用下列代码：

```
bar = new JToolBar(SwingConstants.VERTICAL)
```

或者

```
bar = new JToolBar(titleString, SwingConstants.VERTICAL)
```

按钮是工具栏中最常见的组件类型。然而工具栏中的组件并不仅限如此。例如，可以往工具栏中加入复选框。

9.5.8 工具提示

工具栏有一个缺点，这就是用户常常需要猜测按钮上小图标按钮的含义。为了解决这个问题，用户界面设计者发明了工具提示（tooltips）。当光标停留在某个按钮上片刻时，工具提示就会被激活。工具提示文本显示在一个有颜色的矩形里。当用户移开鼠标时，工具提示就会自动地消失。如图9-28所示。

图9-28 工具提示

在Swing中，可以调用setToolText方法将工具提示添加到JComponent上：

```
exitButton.setToolTipText("Exit");
```

还有一种方法是，如果使用Action对象，就可以用SHORT_DESCRIPTION关联工具提示：

```
exitAction.putValue(Action.SHORT_DESCRIPTION, "Exit");
```

例9-9说明了如何将一个Action对象添加到菜单和工具栏中。注意，动作名在菜单中就是菜单项名，而在工具栏中就是简短的说明。

例9-9 ToolBarTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.13 2007-06-12
7.  * @author Cay Horstmann
8.  */
9. public class ToolBarTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                ToolBarFrame frame = new ToolBarFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame with a toolbar and menu for color changes.
27.  */
28. class ToolBarFrame extends JFrame
29. {
30.    public ToolBarFrame()
31.    {
32.       setTitle("ToolBarTest");
33.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
34.
35.       // add a panel for color change
36.
37.       panel = new JPanel();
38.       add(panel, BorderLayout.CENTER);
39.
40.       // set up actions
41.
42.       Action blueAction = new ColorAction("Blue", new ImageIcon("blue-ball.gif"), Color.BLUE);
43.       Action yellowAction = new ColorAction("Yellow", new ImageIcon("yellow-ball.gif"),
44.             Color.YELLOW);
45.       Action redAction = new ColorAction("Red", new ImageIcon("red-ball.gif"), Color.RED);
46.
47.       Action exitAction = new AbstractAction("Exit", new ImageIcon("exit.gif"))
48.          {
49.             public void actionPerformed(ActionEvent event)
50.             {
51.                System.exit(0);
52.             }
53.          };
54.       exitAction.putValue(Action.SHORT_DESCRIPTION, "Exit");
55.
```

```
56.       // populate tool bar
57.
58.       JToolBar bar = new JToolBar();
59.       bar.add(blueAction);
60.       bar.add(yellowAction);
61.       bar.add(redAction);
62.       bar.addSeparator();
63.       bar.add(exitAction);
64.       add(bar, BorderLayout.NORTH);
65.
66.       // populate menu
67.
68.       JMenu menu = new JMenu("Color");
69.       menu.add(yellowAction);
70.       menu.add(blueAction);
71.       menu.add(redAction);
72.       menu.add(exitAction);
73.       JMenuBar menuBar = new JMenuBar();
74.       menuBar.add(menu);
75.       setJMenuBar(menuBar);
76.    }
77.
78.    public static final int DEFAULT_WIDTH = 300;
79.    public static final int DEFAULT_HEIGHT = 200;
80.
81.    private JPanel panel;
82.
83.    /**
84.     * The color action sets the background of the frame to a given color.
85.     */
86.    class ColorAction extends AbstractAction
87.    {
88.       public ColorAction(String name, Icon icon, Color c)
89.       {
90.          putValue(Action.NAME, name);
91.          putValue(Action.SMALL_ICON, icon);
92.          putValue(Action.SHORT_DESCRIPTION, name + " background");
93.          putValue("Color", c);
94.       }
95.
96.       public void actionPerformed(ActionEvent event)
97.       {
98.          Color c = (Color) getValue("Color");
99.          panel.setBackground(c);
100.      }
101.   }
102. }
```

API javax.swing.JToolBar 1.2

- JToolBar()
- JToolBar(String titleString)
- JToolBar(int orientation)
- JToolBar(String titleString, int orientation)

用给定的标题字符串和方位构造一个工具栏。Orientation可以是SwingConstants.HORIZONTAL（默认）或SwingConstants.VERTICAL。

- `JButton add(Action a)`
  用给定的动作名、图标、简要的说明和动作回调构造一个工具栏中的新按钮。
- `void addSeparator()`
  将一个分隔符添加到工具栏的尾部。

API `javax.swing.JComponent 1.2`

- `void setToolTipText(String text)`
  设置当鼠标停留在组件上时显示在工具提示中的文本。

9.6 复杂的布局管理

迄今为止．在前面的示例应用程序所使用的用户界面组件中，只使用了边框布局、流布局和网格布局。对于复杂的问题而言，只使用这四种布局显然不够。本节将详细地讨论高级布局管理器。

Windows程序员可能会为Java对布局管理器如此兴师动众而感到奇怪。毕竟，在Windows中，布局管理不是一个太大的问题：首先，可以用对话框编辑器将组件拖放到对话柜的表面上。然后，再使用编辑器工具完成组件对齐、均衡间隔、中心定位等工作。如果正在开发的是一个大型项目，可能根本就不必担心组件如何布局，技术娴熟的用户界面设计师会完成所有这些任务。

使用这种方法布局会出现这个的问题：如果组件的大小发生改变，必须手工地更新。为什么组件的大小会发生改变呢？通常有两种可能。第一种可能是为按钮标签和其他对话框文本选择了一种较大的字体。如果在Windows里试验一下就会发现很多应用程序没有解决好这个问题。按钮的尺寸不增大，大字体被紧缩在原来的空间里。当应用程序中的字符串翻译成其他语言时也有可能出现同样的问题，例如，“ Cancel ”在德语中为“ Abbrechen ”。如果一个按钮的大小被设计成刚好能够显示字符串“ Cancel ”，那么德语版显示就会出现问题了，字符串将会被剪掉一部分。

为什么Windows中的按钮不能动态地增大以适应标签呢？这是因为用户界面设计师没有给出应该在哪个方向增大的指令。每个组件拖放或排列之后，对话框编辑器只保存其像素位置和尺寸大小。至于组件为什么以这种方式排列并没有记录下来。

Java布局管理器是一种用于组件布局的好方法。应用布局管理器，布局就可以使用组件间关系的指令来完成布局操作。对于最初的AWT来说，这一点特别重要，这是因为AWT使用的是本地用户界面元素。在Motif、Windows和Macintosh中，按钮和列表框的大小各不相同，而且应用程序或applet不会预先知道它们将在哪个平台上显示。在某种程度上，可变性在Swing中就没有那么重要。如果应用程序强制使用特定的观感，如Metal观感，那么这个程序在所有平台上显示的结果都一样。但是，如果允许应用程序的用户随意地选择观感，则需要依据布局管理器的灵活性调整组件的排列了。

自从Java 1.0以来，AWT就含有网格组布局（grid bag layout），这种布局将组件按行和列排列。行和列的大小可以灵活改变，并且组件可以横跨多行多列。这种布局管理器非常灵活，但也非常复杂。仅仅提及“网格组布局”一词就会吓住一些Java程序员。

Swing设计者有一个失败的尝试：为了能够将程序员从使用网格组布局的困难中解脱出来，提出了一种被称为箱式布局（box layout）的布局管理器。在BoxLayout类的JDK文档中写到：“采用水平和垂直[sic]的不同组合内嵌多个面板将可以获得与GridBagLayout类似的效果，而且降低了复杂度。”然而，由于每个箱子是独立布局的，所以不能使用箱式布局排列水平和垂直方向都相邻的组件。

Java SE1.4还做了一个尝试：设计一种代替网格组组件的布局——（spring layout）。这种布局使用一个虚构的弹簧将同一个容器中的所有组件连接起来。当容器改变大小时，弹簧可以伸展或收缩，以便调节组件的位置。这听起来似乎感觉枯燥且不可思义，其实也确实如此。弹簧布局很快就会陷入含糊不清的境地。

在2005年，NetBeans开发队伍发明了Matisse技术，这种技术将布局工具与布局管理器结合起来。用户界面设计者可以使用工具将组件拖拽到容器中，并指出组件的排列方式。工具将设计者的意图转换成组布局管理器的可以理解的指令，与手工地编写布局管理的代码相比，这样做要便捷得多。组布局管理器是Java SE 6中的一个新特性。即使没有用NetBeans作为IDE，也应该考虑使用它的GUI建造工具。可以用NetBeans设计GUI，然后再将得到的代码粘贴到所选择的IDE中。

接下来，将讲述网格组布局。这是因为这种布局在早期的Java版本中，使用的最普遍，且也是产生布局代码的最简单方式。下面的策略可以让网格组布局的使用相对简单些。

随后，介绍Matisse工具和组布局管理器。我们需要了解组布局管理器的工作过程，以便能够在用可视化方式将组件放置在某个位置时能够查看Matisse记录的指令是否正确。

最后，将演示如何完全不使用布局管理，手工地放置组件，以及如何编写自己的布局管理器。

9.6.1 网格组布局

网格组布局是所有布局管理器之首。可以将网格组布局看成是没有任何限制的网格布局。在网格组布局中，行和列的尺寸可以改变。可以将相邻的单元合并以适应较大的组件（很多字处理器以及HTML都利用这个功能编辑表格：一旦需要就合并相邻的单元格）。组件不需要填充整个单元格区域，并可以指定它们在单元格内的对齐方式。

请看图9-29中所示的字体选择对话框，其中包含下面的组件：

- 两个用于指定字体外观和大小的组合框
- 两个组合框的标签
- 两个用于选择粗体和斜体的复选框
- 一个用于显示示例字符串的本文区

图9-29 字体对话框

现在，将容器分解为网格单元，如图9-29所示（行和列的

尺寸不需要相同)。每个复选框横跨两列，文本区横跨四行。

要想使用网格组管理器进行布局，必须经过下列过程：

1）建立一个GridBagLayout的对象。不需要指定网格的行数和列数。布局管理器会根据后面所给的信息猜测出来。

2）将GridBagLayout对象设置成组件的布局管理器。

3）为每个组件建立一个GridBagConstraints对象。设置GridBagConstraints对象的域以便指出组件在网格组中的布局方案。

图9-30 设计中使用的对话框网格

4）最后，通过下面的调用添加组件的约束：

```
add(component, constraints);
```

下面给出了相应的代码（稍后将更加详细地介绍各种约束。如果现在不明白约束的作用，不必担心)。

```
GridBagLayout layout = new GridBagLayout();
panel.setLayout(layout);
GridBagConstraints constraints = new GridBagConstraints();
constraints.weightx = 100;
constraints.weighty = 100;
constraints.gridx = 0;
constraints.gridy = 2;
constraints.gridwidth = 2;
constraints.gridheight = 1;
panel.add(component, constraints);
```

知道如何设置GridBagConstraints对象的状态是非常重要的。下面将详细地介绍几个最重要的约束。

1. gridx、gridy、gridwidth和 gridheight 参数

这些约束定义了组件在网格中的位置。gridx和gridy指定了被添加组件左上角的行、列位置。gridwidth和gridheight指定了组件占据的行数和列数。

网格的坐标从0开始。gridx=0和gridy=0代表最左上角。例如，示例程序中，文本区的gridx=2，gridy=0。这是因为这个文本区起始于0行2列（即第3列)，girdwidth=1,gridheight=4因为它横跨了4行1列。

2. 增量域

在网格布局中，需要为每个区域设置增量域（weightx和weighty)。如果将增量设置为0，则这个区域将永远为初始尺寸。在如图9-29所示的网格布局中，由于将标签的weightx设置为0，所以在窗口缩放时，标签大小始终保持不变。另一方面，如果将所有区域的增量都设置为0，容器就会集聚在为它分配的区域中间，而不是通过拉伸来填充它。

从概念上讲，增量参数属于行和列的属性，而不属于某个单独的单元格。但却需要在单元

格上指定它们，这是因为网格组布局并不暴露行和列。行和列的增量等于每行或每列单元格的增量最大值。因此，如果想让一行或一列的大小保持不变，就需要将这行、这列的所有组件的增量都设置为0。

注意，增量并不实际给出列的相对大小。当容器超过首选大小时，增量表示分配给每个区域的扩展比例值。这么说并不太直观。这里建议将所有的增量设置为100，运行程序，查看一下布局情况。缩放对话框，查看一下行和列是如何调整的。如果发现某行或某列不应该扩大，就将那行或那列中的所有组件的增量设置为0。也可以使用其他的增量值进行修补，但是这么做的意义并不大。

3. fill和anchor参数

如果不希望组件拉伸至整个区域，就需要设置fill约束。它有四个有效值：GridBagConstraints.NONE、GridBagConstraints.HORIZONTAL、GridBagConstraints. VERTICAL和GridBagConstraints.BOTH.。

如果组件没有填充整个区域，可以通过设置anchor域指定其位置。有效值为GridBagConstraints.CENTER(默认值)、GridBagConstraints.NORTH、GridBagConstraints. NORTHEAST和GridBagConstraints.EAST等。

4. 填塞

可以通过设置GridBagLayout的insets域在组件周围增加附加的空白区域。通过设置Insets对象的left、top、right 和bottom指定组件周围的空间量。这被称作外部填塞（external padding)。

通过设置ipadx和ipady指定内部填塞（internal padding)。这两个值被加到组件的最小宽度和最小高度上。这样可以保证组件不会收缩至最小尺寸之下。

5. 指定gridx, gridy, gridwidth和gridheight参数的另一种方法

AWT文档建议不要将gridx和gridy设置为绝对位置，应该将它们设置为常量GridBagConstraints.RELATIVE。然后，按照标准的顺序，将组件添加到网格组布局中。即第一行从左向右，然后再开始新的一行，以此类推。

还需要通过为gridheight和gridwidth域指定一个适当的值来设置组件横跨的行数和列数。除此之外，如果组件扩展至最后一行或最后一列，则不要给出一个实际的数值，而是用常量GridBagConstraints.REMAINDER替代，这样会告诉布局管理器这个组件是本行上的最后一个组件。

这种方案看起来能起作用，但似乎显得有点笨拙。这是因为这样做会将实际位置信息对布局管理器隐藏起来，而日后又希望它能够重新发现这些信息。

这些事情看起来都很麻烦和复杂，但实际上，下面的策略可以让网格组布局的使用相对简单一些：

1）在纸上画出组件布局草图。

2）找出一种网格，小组件被放置在一个单元格内，大组件将横跨多个单元格。

3）用0，1，2……标识网格的行和列。现在可以读取gridx, gridy, gridwidth和gridheight的值。

4）对于每个组件，需要考虑下列问题：是否需要水平或者垂直填充它所在的单元格？如果不需要，希望如何排列？这些就是fill和anchor参数的设置。

5）将所有的增量设置为100。如果需要某行或某列始终保持默认的大小，就将这行或这列中所有组件的weightx和weighty设置为0。

6）编写代码。仔细地检查GridBagConstraints的设置。错误的约束可能会破坏整个布局。

7）编译、运行。

有些GUI构造者使用工具来可视化地指定约束。如图9-31就是NetBeans中的配置对话框。

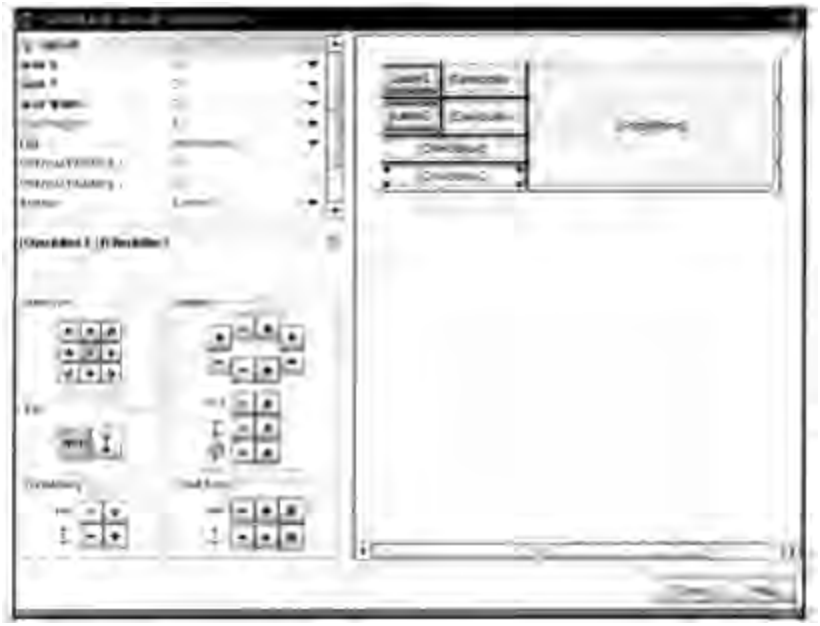


图9-31　在NetBeans中指定网格组的约束

6. 使用帮助类来管理（tame）网格组约束

网格组布局最乏味的工作就是为设置约束编写代码。为此，很多程序员编写帮助函数或者帮助类来满足上面的目的。下面是为字体对话框示例编写的帮助类。这个类有下列特性：

- 名字简短：GBC代替GridBagConstraints。
- 扩展于GridBagConstraints，因此可以使用约束的缩写，如GBC.EAST。
- 当添加组件时，使用GBC对象，如：

```
add(component, new GBC(1, 2));
```

- 有两个构造器可以用来设置最常用的参数：gridx和gridy，或者gridx、gridy、gridwidth和gridheight。

```
add(component, new GBC(1, 2, 1, 4));
```

- 域有很便捷的设置器setter，x/y值对：

```
add(component, new GBC(1, 2).setWeight(100, 100));
```

- Setter方法将返回this，所以可以链接它们：

```
add(component, new GBC(1, 2).setAnchor(GBC.EAST).setWeight(100, 100));
```

• setInsets方法将构造Inset对象。要想获取1个像素的insets，可以调用：

```
add(component, new GBC(1, 2).setAnchor(GBC.EAST).setInsets(1));
```

例9-10显示了字体对话框示例的全部代码。下面是将组件添加到网格组中的代码：

```
add(faceLabel, new GBC(0, 0).setAnchor(GBC.EAST));
add(face, new GBC(1, 0).setFill(GBC.HORIZONTAL).setWeight(100, 0).setInsets(1));
add(sizeLabel, new GBC(0, 1).setAnchor(GBC.EAST));
add(size, new GBC(1, 1).setFill(GBC.HORIZONTAL).setWeight(100, 0).setInsets(1));
add(bold, new GBC(0, 2, 2, 1).setAnchor(GBC.CENTER).setWeight(100, 100));
add(italic, new GBC(0, 3, 2, 1).setAnchor(GBC.CENTER).setWeight(100, 100));
add(sample, new GBC(2, 0, 1, 4).setFill(GBC.BOTH).setWeight(100, 100));
```

一旦理解了网格组约束，就会觉得这些代码十分易于阅读且便于调试。

注释：Sun在http://java.sun.com/docs/books/tutorial/uiswing/layout/gridbag.html的指南中建议：对于所有的组件重用同一个GridBagConstraints对象。我们发现这样做将会使代码难于阅读并易于发生错误。例如，请看http://java.sun.com/docs/books/tutorial/uiswing/events/containerlistener.html的演示。按钮真的被水平拉伸吗？还是程序员忘记了关闭fill约束设定的BOTH？

例9-10 GridBagLayoutTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.33 2007-06-12
7.  * @author Cay Horstmann
8.  */
9. public class GridBagLayoutTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                FontFrame frame = new FontFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame that uses a grid bag layout to arrange font selection components.
27.  */
28. class FontFrame extends JFrame
29. {
30.    public FontFrame()
31.    {
32.       setTitle("GridBagLayoutTest");
33.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
```

```
34.
35.       GridBagLayout layout = new GridBagLayout();
36.       setLayout(layout);
37.
38.       ActionListener listener = new FontAction();
39.
40.       // construct components
41.
42.       JLabel faceLabel = new JLabel("Face: ");
43.
44.       face = new JComboBox(new String[] { "Serif", "SansSerif", "Monospaced", "Dialog",
45.             "DialogInput" });
46.
47.       face.addActionListener(listener);
48.
49.       JLabel sizeLabel = new JLabel("Size: ");
50.
51.       size = new JComboBox(new String[] { "8", "10", "12", "15", "18", "24", "36", "48" });
52.
53.       size.addActionListener(listener);
54.
55.       bold = new JCheckBox("Bold");
56.       bold.addActionListener(listener);
57.
58.       italic = new JCheckBox("Italic");
59.       italic.addActionListener(listener);
60.
61.       sample = new JTextArea();
62.       sample.setText("The quick brown fox jumps over the lazy dog");
63.       sample.setEditable(false);
64.       sample.setLineWrap(true);
65.       sample.setBorder(BorderFactory.createEtchedBorder());
66.
67.       // add components to grid, using GBC convenience class
68.
69.       add(faceLabel, new GBC(0, 0).setAnchor(GBC.EAST));
70.       add(face, new GBC(1, 0).setFill(GBC.HORIZONTAL).setWeight(100, 0).setInsets(1));
71.       add(sizeLabel, new GBC(0, 1).setAnchor(GBC.EAST));
72.       add(size, new GBC(1, 1).setFill(GBC.HORIZONTAL).setWeight(100, 0).setInsets(1));
73.       add(bold, new GBC(0, 2, 2, 1).setAnchor(GBC.CENTER).setWeight(100, 100));
74.       add(italic, new GBC(0, 3, 2, 1).setAnchor(GBC.CENTER).setWeight(100, 100));
75.       add(sample, new GBC(2, 0, 1, 4).setFill(GBC.BOTH).setWeight(100, 100));
76.    }
77.
78.    public static final int DEFAULT_WIDTH = 300;
79.    public static final int DEFAULT_HEIGHT = 200;
80.
81.    private JComboBox face;
82.    private JComboBox size;
83.    private JCheckBox bold;
84.    private JCheckBox italic;
85.    private JTextArea sample;
86.
87.    /**
88.     * An action listener that changes the font of the sample text.
89.     */
90.    private class FontAction implements ActionListener
91.    {
```

```
92.       public void actionPerformed(ActionEvent event)
93.       {
94.          String fontFace = (String) face.getSelectedItem();
95.          int fontStyle = (bold.isSelected() ? Font.BOLD : 0)
96.                + (italic.isSelected() ? Font.ITALIC : 0);
97.          int fontSize = Integer.parseInt((String) size.getSelectedItem());
98.          Font font = new Font(fontFace, fontStyle, fontSize);
99.          sample.setFont(font);
100.         sample.repaint();
101.      }
102.   }
103. }
```

例9-11显示了CBC帮助类的代码。

例9-11 GBC.java

```
1. import java.awt.*;
2.
3. /**
4.  * This class simplifies the use of the GridBagConstraints class.
5.  * @version 1.01 2004-05-06
6.  * @author Cay Horstmann
7.  */
8. public class GBC extends GridBagConstraints
9. {
10.   /**
11.    * Constructs a GBC with a given gridx and gridy position and all other grid
12.    * bag constraint values set to the default.
13.    * @param gridx the gridx position
14.    * @param gridy the gridy position
15.    */
16.   public GBC(int gridx, int gridy)
17.   {
18.      this.gridx = gridx;
19.      this.gridy = gridy;
20.   }
21.
22.   /**
23.    * Constructs a GBC with given gridx, gridy, gridwidth, gridheight and all
24.    * other grid bag constraint values set to the default.
25.    * @param gridx the gridx position
26.    * @param gridy the gridy position
27.    * @param gridwidth the cell span in x-direction
28.    * @param gridheight the cell span in y-direction
29.    */
30.   public GBC(int gridx, int gridy, int gridwidth, int gridheight)
31.   {
32.      this.gridx = gridx;
33.      this.gridy = gridy;
34.      this.gridwidth = gridwidth;
35.      this.gridheight = gridheight;
36.   }
37.
38.   /**
39.    * Sets the anchor.
```

```
40.    * @param anchor the anchor value
41.    * @return this object for further modification
42.    */
43.   public GBC setAnchor(int anchor)
44.   {
45.      this.anchor = anchor;
46.      return this;
47.   }
48.
49.   /**
50.    * Sets the fill direction.
51.    * @param fill the fill direction
52.    * @return this object for further modification
53.    */
54.   public GBC setFill(int fill)
55.   {
56.      this.fill = fill;
57.      return this;
58.   }
59.
60.   /**
61.    * Sets the cell weights.
62.    * @param weightx the cell weight in x-direction
63.    * @param weighty the cell weight in y-direction
64.    * @return this object for further modification
65.    */
66.   public GBC setWeight(double weightx, double weighty)
67.   {
68.      this.weightx = weightx;
69.      this.weighty = weighty;
70.      return this;
71.   }
72.
73.   /**
74.    * Sets the insets of this cell.
75.    * @param distance the spacing to use in all directions
76.    * @return this object for further modification
77.    */
78.   public GBC setInsets(int distance)
79.   {
80.      this.insets = new Insets(distance, distance, distance, distance);
81.      return this;
82.   }
83.
84.   /**
85.    * Sets the insets of this cell.
86.    * @param top the spacing to use on top
87.    * @param left the spacing to use to the left
88.    * @param bottom the spacing to use on the bottom
89.    * @param right the spacing to use to the right
90.    * @return this object for further modification
91.    */
92.   public GBC setInsets(int top, int left, int bottom, int right)
93.   {
94.      this.insets = new Insets(top, left, bottom, right);
95.      return this;
96.   }
```

```
97.
98.    /**
99.     * Sets the internal padding
100.    * @param ipadx the internal padding in x-direction
101.    * @param ipady the internal padding in y-direction
102.    * @return this object for further modification
103.    */
104.   public GBC setIpad(int ipadx, int ipady)
105.   {
106.      this.ipadx = ipadx;
107.      this.ipady = ipady;
108.      return this;
109.   }
110. }
```

API java.awt.GridBagConstraints 1.0

- `int gridx, gridy`
  指定单元格的起始行和列。默认值为0。
- `int gridwidth, gridheight`
  指定单元格的行和列的范围。默认值为1。
- `double weightx, weighty`
  指定单元格扩大时的容量。默认值为0。
- `int anchor`
  表示组件在单元格内的对齐方式。可以选择的绝对位置有：

NORTHWEST	NORTH	NORTHEAST
WEST	CENTER	EAST
SOUTHWEST	SOUTH	SOUTHEAST

  或者各个方向上的相对位置：

FIRST_LINE_START	LINE_START	FIRST_LINE_END
PAGE_START	CENTER	PAGE_END
LAST_LINE_START	LINE_END	LAST_LINE_END

  如果应用程序有可能从右向左，或者自顶至底排列文本，就应该使用后者。默认值为CENTER。
- `int fill`
  指定组件在单元格内的填充行为，取值为NONE, BOTH, HORIZONTAL，或者VERTICAL。默认值为NONE。
- `int ipadx, ipady`
  指定组件周围的“内部”填充。默认值为0。
- `Insets insets`
  指定组件边框周围的“外部”填充。默认为不填充。
- `GridBagConstraints(int gridx, int gridy, int gridwidth, int gridheight, double`

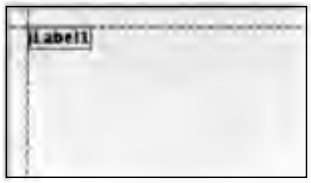
`weightx, double weighty, int anchor, int fill, Insets insets, int ipadx, int ipady) 1.2`

用参数中给定的所有域值构造GridBagConstraints。Sun建议这个构造器只用在自动代码生成器中，因为它会使得代码非常难于阅读。

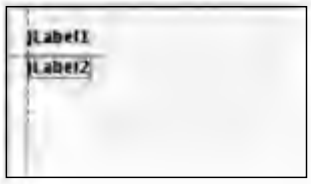
9.6.2 组布局

在讨论GroupLayout类的API之前，先快速地浏览一下NetBeans中的Matisse GUI构造器。这里并没有讲述Matisse全部的使用方法，有关更加详细的信息请参看网站http://netbeans.org/kb/articles/matisse.html。

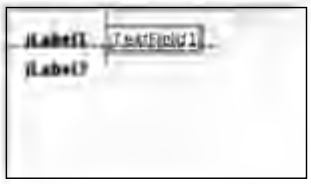
下面是布局如图9-13所示的对话框的工作流程。创建一个新项目，并添加一个新的JFrame表单。拖拽一个标签，直到出现了两条分离于容器边框的引导线：



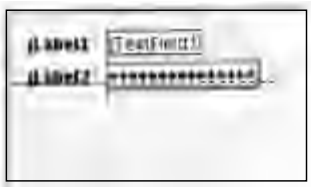
在第一行下面放置另一个标签：



拖拽一个文本域，让文本域的基线与第一个标签的基线对齐。再次注意引导线：



最后，放置一个密码域，它的左侧是标签，上方是文本域。



Matisse将这些操作转换成下列Java代码：

```
layout.setHorizontalGroup(
    layout.createParallelGroup(GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addContainerGap()
```

```
        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jLabel1)
                .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jTextField1))
            .addGroup(layout.createSequentialGroup()
                .addComponent(jLabel2)
                .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jPasswordField1)))
        .addContainerGap(222, Short.MAX_VALUE)));
layout.setVerticalGroup(
    layout.createParallelGroup(GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
            .addComponent(jTextField1))
        .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
            .addComponent(jPasswordField1))
        .addContainerGap(244, Short.MAX_VALUE)));
```

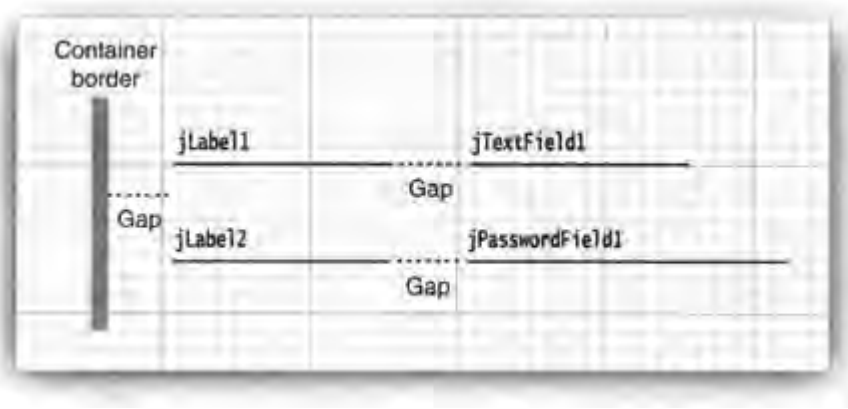
看起来有点让人感觉惊恐，庆幸的是不需要编写代码。阅读一下这段代码会有助于理解布局行为的操作过程，以便能够发现程序中的错误。下面分析一下这段代码的基本结构。在本节后面有关API注解中将更加详细地解释每个类和方法。

可以通过将组件放入GroupLyout.SequentialGroup或者GroupLayout.ParallelGroup对象中将它们组织起来。这些类是GroupLayout.Group的子类。在组中可以包含组件、间距和内嵌的组。由于组类中的各种add方法都返回组对象，因此可以像下面这样将方法调用串联在一起：

```
group.addComponent(...).addPreferredGap(...).addComponent(...);
```

正像从示例代码中所看到的，组布局分别对水平和垂直布局进行计算。

为了能够看到水平计算的效果，假设组件都被拍平了，因此高度为0，如下所示：



有两个平行的组件序列，对应的代码（略有简化）是：

```
.addContainerGap()
.addGroup(layout.createParallelGroup()
    .addGroup(layout.createSequentialGroup()
```

```
            .addComponent(jLabel1)
            .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jTextField1))
        .addGroup(layout.createSequentialGroup()
            .addComponent(jLabel2)
            .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jPasswordField1)))
```

但是，请稍等，上面这段代码有问题。如果两个标签的长度不一样，那么文本域和密码域就无法对齐。

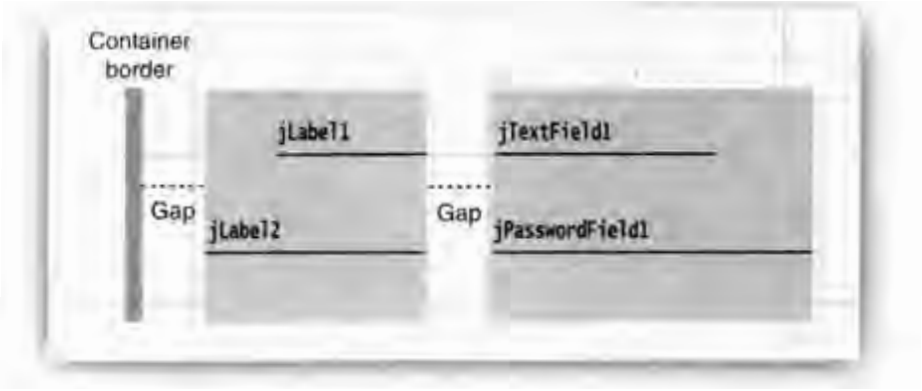
必须通告Matisse，这里希望将组件对齐。选择这两个域，然后点击鼠标右键，并从菜单中选择Align→Left to Column。同样可以对齐标签。如图9-32所示。

图9-32 在Matisse中对齐标签和文本域

这样做后，戏曲性地改变了布局代码：

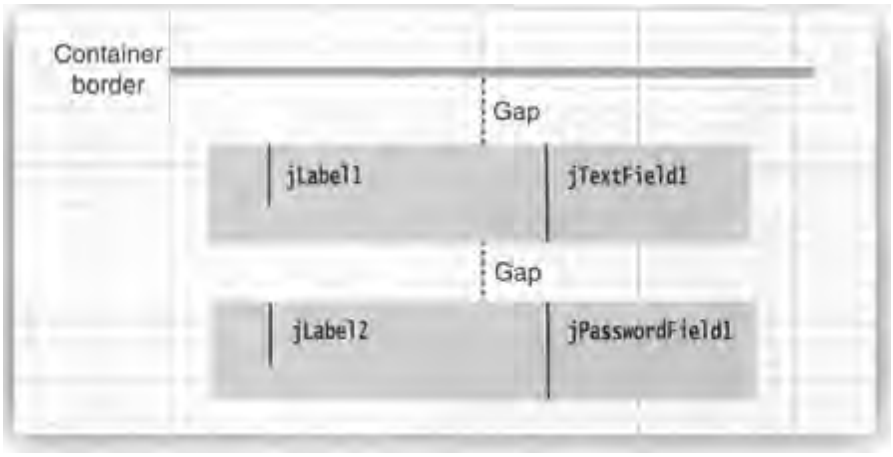
```
.addGroup(layout.createSequentialGroup()
    .addContainerGap()
    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
        .addComponent(jLabel1, GroupLayout.Alignment.TRAILING)
        .addComponent(jLabel2, GroupLayout.Alignment.TRAILING))
    .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
        .addComponent(jTextField1)
        .addComponent(jPasswordField1))
```

现在，标签和域分别置于两个平行组中。第一组的对齐方式是TRAILING（这意味着当文本方向是自左向右时，应该右对齐）



简直太奇妙了！Matisse居然可以将设计者的指令转换成嵌套的组。其实，正如Arthur C.Clarke所说：任何高级技术都源于奇特的想法之中。

鉴于完整性的考虑，下面看一下垂直计算。此时，应该将组件看作没有宽度。这里有一个顺序排列的组，其中包含了两个用间距分隔的平行组：



对应的代码是：

```
layout.createSequentialGroup()
    .addContainerGap()
    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel1)
        .addComponent(jTextField1))
    .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel2)
        .addComponent(jPasswordField1))
```

从这段代码可以看到：组件依据基线对齐（基线是组件文本对齐的直线）。

注释：在Java较早的版本中，不可能实现基线的精确对齐。在Java SE6的Component类中增加了一个方法getBaseline，以便能够精确地获得含有文本的组件的基线。

可以将一组组件的大小强制为相等。例如，可能想确保文本域和密码域的宽度相等。在Matisse中，选择这两个组件，然后点击鼠标右键，并从菜单中Same Size→Same Width。如图9-33所示。

Matisse将下面这条语句添加到布局代码中：

```
layout.linkSize(SwingConstants.HORIZONTAL, new Component[] {jPasswordField1, jTextField1});
```

在例9-12的代码中显示了如何用GroupLayout替代GridBagLayout来布局前面讲述的字体选择器。这里的代码看起来可能比例9-10要复杂些，但并不必编写。可以使用Matisse进行布局，然后，对少量的代码整理一下即可。

图9-33 强制两个组件的宽度相等

例9-12 GroupLayoutTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.0 2007-04-27
7.  * @author Cay Horstmann
8.  */
9. public class GroupLayoutTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                FontFrame frame = new FontFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame that uses a group layout to arrange font selection components.
27.  */
28. class FontFrame extends JFrame
29. {
30.    public FontFrame()
31.    {
32.       setTitle("GroupLayoutTest");
33.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
34.
35.       ActionListener listener = new FontAction();
36.
37.       // construct components
```

```
38.
39.       JLabel faceLabel = new JLabel("Face: ");
40.
41.       face = new JComboBox(new String[] { "Serif", "SansSerif", "Monospaced", "Dialog",
42.             "DialogInput" });
43.
44.       face.addActionListener(listener);
45.
46.       JLabel sizeLabel = new JLabel("Size: ");
47.
48.       size = new JComboBox(new String[] { "8", "10", "12", "15", "18", "24", "36", "48" });
49.
50.       size.addActionListener(listener);
51.
52.       bold = new JCheckBox("Bold");
53.       bold.addActionListener(listener);
54.
55.       italic = new JCheckBox("Italic");
56.       italic.addActionListener(listener);
57.
58.       sample = new JTextArea();
59.       sample.setText("The quick brown fox jumps over the lazy dog");
60.       sample.setEditable(false);
61.       sample.setLineWrap(true);
62.       sample.setBorder(BorderFactory.createEtchedBorder());
63.
64.       pane = new JScrollPane(sample);
65.
66.       GroupLayout layout = new GroupLayout(getContentPane());
67.       setLayout(layout);
68.       layout.setHorizontalGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
69.             .addGroup(
70.                   layout.createSequentialGroup().addContainerGap().addGroup(
71.                         layout.createParallelGroup(GroupLayout.Alignment.LEADING).addGroup(
72.                               GroupLayout.Alignment.TRAILING,
73.                               layout.createSequentialGroup().addGroup(
74.                                     layout.createParallelGroup(GroupLayout.Alignment.TRAILING)
75.                                           .addComponent(faceLabel).addComponent(sizeLabel))
76.                                     .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
77.                                     .addGroup(
78.                                           layout.createParallelGroup(
79.                                                 GroupLayout.Alignment.LEADING, false)
80.                                                 .addComponent(size).addComponent(face)))
81.                               .addComponent(italic).addComponent(bold)).addPreferredGap(
82.                         LayoutStyle.ComponentPlacement.RELATED).addComponent(pane)
83.                         .addContainerGap())));
84.
85.       layout.linkSize(SwingConstants.HORIZONTAL, new java.awt.Component[] { face, size });
86.
87.       layout.setVerticalGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
88.             .addGroup(
89.                   layout.createSequentialGroup().addContainerGap().addGroup(
90.                         layout.createParallelGroup(GroupLayout.Alignment.LEADING).addComponent(
91.                               pane, GroupLayout.Alignment.TRAILING).addGroup(
92.                               layout.createSequentialGroup().addGroup(
93.                                     layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
94.                                           .addComponent(face).addComponent(faceLabel))
```

```
                                    .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                                    .addGroup(
                                          layout.createParallelGroup(
                                            GroupLayout.Alignment.BASELINE).addComponent(size)
                                                .addComponent(sizeLabel)).addPreferredGap(
                                        LayoutStyle.ComponentPlacement.RELATED).addComponent(
                                          italic, GroupLayout.DEFAULT_SIZE,
                                          GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                    .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                                    .addComponent(bold, GroupLayout.DEFAULT_SIZE,
                                          GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                        .addContainerGap()));
   }

   public static final int DEFAULT_WIDTH = 300;
   public static final int DEFAULT_HEIGHT = 200;

   private JComboBox face;
   private JComboBox size;
   private JCheckBox bold;
   private JCheckBox italic;
   private JScrollPane pane;
   private JTextArea sample;

   /**
    * An action listener that changes the font of the sample text.
    */
   private class FontAction implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         String fontFace = (String) face.getSelectedItem();
         int fontStyle = (bold.isSelected() ? Font.BOLD : 0)
               + (italic.isSelected() ? Font.ITALIC : 0);
         int fontSize = Integer.parseInt((String) size.getSelectedItem());
         Font font = new Font(fontFace, fontStyle, fontSize);
         sample.setFont(font);
         sample.repaint();
      }
   }
}
```

API javax.swing.GroupLayout 6

- `GroupLayout(Container host)`
  构造一个GroupLayout对象，用于布局host容器中的组件（注意： host容器仍然需要调用setLayout）。
- `void setHorizontalGroup(GroupLayout.Group g)`
- `void setVerticalGroup(GroupLayout.Group g)`
  设置用于控制水平或垂直容器的组。
- `void linkSize(Component... components)`
- `void linkSize(int axis, Component... component)`

强制给定的几个组件具有相同的尺寸，或者在指定的坐标轴上有相同的尺寸（SwingConstants.HORIZONTAL或者SwingContants.VERTICAL）。

- `GroupLayout.SequentialGroup createSequentialGroup()`

创建一个组，用于平行地布局子组件。

- `GroupLayout.ParallelGroup createParallelGroup()`
- `GroupLayout.ParallelGroup createParallelGroup(GroupLayout.Alignment align)`
- `GroupLayout.ParallelGroup createParallelGroup(GroupLayout.Alignment align,boolean resizable)`

创建一个组，用于顺序地布局子组件。

参数：align　BASELINE、LEADING（默认值）、TRAILING或CENTER

resizable　如果组可以调整大小，这个值为true（默认值）；如果首选的尺寸是最小尺寸或最大尺寸，这个值为false

- `boolean getHonorsvisibility()`
- `void setHonorsvisibility(boolean b)`

获取或设置honorsVisibility特性。当这个值为true（默认值）时，不可见的组件不参与布局。当这个值为false时，好像可见的组件一样，这些组件也参与布局。这个特性主要用于想要临时隐藏一些组件，而又不希望改变布局的情况。

- `boolean getAutoCreateCaps()`
- `void setAutoCreateCaps(boolean b)`
- `boolean getAutoCreateContainerCaps()`
- `void setAutoCreateContainerCaps(boolean b)`

获取或设置autoCreateCaps和autoCreateContainerCaps特性。当这个值为true时，将自动地在组件或容器边框之间添加空隙。默认值是false。在手工地构造GroupLayout时，true值很有用。

API `javax.swing.GroupLayout.Group`

- `GroupLayout.Group addComponent(Component c)`
- `GroupLayout.Group addComponent(Component c, int minimumSize, int preferredSize, int maximumSize)`

添加一个组件至本组中。尺寸参数可以是一个实际值（非负值），或者是一个特定的常量GroupLayout.DEFAULT_SIZE或GroupLayout.PREFERRED_SIZE。当使用DEFAULT_SIZE，将调用组件的getMinimumSize、getPreferredSize或getMaximumSize。当使用PREFERRED_SIZE时，将调用getPreferredSize方法。

- `GroupLayout.Group addCap(int size)`
- `GroupLayout.Group addCap(int minimumSize, int preferredSize, int maximumSize)`

添加一个固定的或可调节的间隙。

- `GroupLayout.Group addGroup(GroupLayout.Group g)`

将一个给定的组添加到本组中。

API javax.swing.GroupLayout.ParallelGroup

- GroupLayout.ParallelGroup addComponent(Component c, GroupLayout.Alignment align)
- GroupLayout.ParallelGroup addComponent(Component c, GroupLayout.Alignment align, int minimumSize, int preferredSize, int maximumSize)
- GroupLayout.ParallelGroup addGroup(GroupLayout.Group g, GroupLayout.Alignment align)

  利用给定的对齐方式（BASELINE、LEADING、TRAILINC或CENTER）添加一个组件或组至本组中。

API javax.swing.GroupLayout.SequentialGroup

- GroupLayout.SequentialGroup addContainerCap()
- GroupLayout.SequentialGroup addContainerCap(int preferredSize, int maximumSize)

  为分隔组件和容器的边缘添加一个间隙。
- GroupLayout.SequentialGroup addPreferredCap(LayoutStyle.ComponentPlacement type)

  为分隔组件添加一个间隙。间隙的类型是LayoutStyle.ComponentPlacement.RELATED或LayoutStyle.ComponentPlacement。

9.6.3 不使用布局管理器

有时候用户可能不想使用任何布局管理器，而只想把组件放在一个固定的位置上（通常称为绝对定位）。这对于与平台无关的应用程序来说并不是一个好主意，但可用来快速地构造原型。

下面是将一个组件定位到某个绝对定位的步骤：

1）将布局管理器设置为null。

2）将组件添加到容器中。

3）指定想要放置的位置和大小。

```
frame.setLayout(null);
JButton ok = new JButton("Ok");
frame.add(ok);
ok.setBounds(10, 10, 30, 15);
```

API java.awt.Component 1.0

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

  移动并调节组件的尺寸

  参数：x, y　　组件新的左上角位置

width, height 组件新的尺寸

9.6.4 定制布局管理器

原则上，可以通过自己设计LayoutManager类来实现特殊的布局方式。例如，可以将容器中的组件排列成一个圆形。要想达到这个效果，通常很消耗间和精力，但是如图9-34所示，结果可能非常好看。

图9-34 圆形布局

如果一定需要定制布局管理器，可以采用下面讲述的定制布局管理器的方法。定制布局管理器必须实现LayoutManager接口，并且需要定义下面5个方法：

```
void addLayoutComponent(String s, Component c);
void removeLayoutComponent(Component c);
Dimension preferredLayoutSize(Container parent);
Dimension minimumLayoutSize(Container parent);
void layoutContainer(Container parent);
```

在添加或删除一个组件时会调用前面两个方法。如果不需要保存组件的任何附加信息，那么可以将这两个方法置为空。接下来的两个方法计算组件的最小布局和首选布局所需要的空间。两者通常相等。第5个方法真正地实施操作，它调用所有组件的setBounds方法。

注释：AWT还有第二个接口LayoutManager2，其中包含10个需要实现的方法，而不是5个。这个接口的主要特点是允许用户使用带有约束的add方法。例如，BorderLayout和GridBagLayout都实现了LayoutManager2接口。

例9-13简单实现了CircleLayout管理器，在父组件中沿着圆形排列组件。这个管理器很有趣，但是没有什么实际的应用价值。

例9-13 CircleLayoutTest.java

```
1. import java.awt.*;
2.
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.32 2007-06-12
7.  * @author Cay Horstmann
8.  */
9. public class CircleLayoutTest
10. {
```

```
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                CircleLayoutFrame frame = new CircleLayoutFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame that shows buttons arranged along a circle.
27.  */
28. class CircleLayoutFrame extends JFrame
29. {
30.    public CircleLayoutFrame()
31.    {
32.       setTitle("CircleLayoutTest");
33.
34.       setLayout(new CircleLayout());
35.       add(new JButton("Yellow"));
36.       add(new JButton("Blue"));
37.       add(new JButton("Red"));
38.       add(new JButton("Green"));
39.       add(new JButton("Orange"));
40.       add(new JButton("Fuchsia"));
41.       add(new JButton("Indigo"));
42.       pack();
43.    }
44. }
45.
46. /**
47.  * A layout manager that lays out components along a circle.
48.  */
49. class CircleLayout implements LayoutManager
50. {
51.    public void addLayoutComponent(String name, Component comp)
52.    {
53.    }
54.
55.    public void removeLayoutComponent(Component comp)
56.    {
57.    }
58.
59.    public void setSizes(Container parent)
60.    {
61.       if (sizesSet) return;
62.       int n = parent.getComponentCount();
63.
64.       preferredWidth = 0;
65.       preferredHeight = 0;
66.       minWidth = 0;
67.       minHeight = 0;
```

```
68.       maxComponentWidth = 0;
69.       maxComponentHeight = 0;
70.
71.       // compute the maximum component widths and heights
72.       // and set the preferred size to the sum of the component sizes.
73.       for (int i = 0; i < n; i++)
74.       {
75.          Component c = parent.getComponent(i);
76.          if (c.isVisible())
77.          {
78.             Dimension d = c.getPreferredSize();
79.             maxComponentWidth = Math.max(maxComponentWidth, d.width);
80.             maxComponentHeight = Math.max(maxComponentHeight, d.height);
81.             preferredWidth += d.width;
82.             preferredHeight += d.height;
83.          }
84.       }
85.       minWidth = preferredWidth / 2;
86.       minHeight = preferredHeight / 2;
87.       sizesSet = true;
88.    }
89.
90.    public Dimension preferredLayoutSize(Container parent)
91.    {
92.       setSizes(parent);
93.       Insets insets = parent.getInsets();
94.       int width = preferredWidth + insets.left + insets.right;
95.       int height = preferredHeight + insets.top + insets.bottom;
96.       return new Dimension(width, height);
97.    }
98.
99.    public Dimension minimumLayoutSize(Container parent)
100.   {
101.      setSizes(parent);
102.      Insets insets = parent.getInsets();
103.      int width = minWidth + insets.left + insets.right;
104.      int height = minHeight + insets.top + insets.bottom;
105.      return new Dimension(width, height);
106.   }
107.
108.   public void layoutContainer(Container parent)
109.   {
110.      setSizes(parent);
111.
112.      // compute center of the circle
113.
114.      Insets insets = parent.getInsets();
115.      int containerWidth = parent.getSize().width - insets.left - insets.right;
116.      int containerHeight = parent.getSize().height - insets.top - insets.bottom;
117.
118.      int xcenter = insets.left + containerWidth / 2;
119.      int ycenter = insets.top + containerHeight / 2;
120.
121.      // compute radius of the circle
122.
123.      int xradius = (containerWidth - maxComponentWidth) / 2;
124.      int yradius = (containerHeight - maxComponentHeight) / 2;
```

```
125.       int radius = Math.min(xradius, yradius);
126.
127.       // lay out components along the circle
128.
129.       int n = parent.getComponentCount();
130.       for (int i = 0; i < n; i++)
131.       {
132.          Component c = parent.getComponent(i);
133.          if (c.isVisible())
134.          {
135.             double angle = 2 * Math.PI * i / n;
136.
137.             // center point of component
138.             int x = xcenter + (int) (Math.cos(angle) * radius);
139.             int y = ycenter + (int) (Math.sin(angle) * radius);
140.
141.             // move component so that its center is (x, y)
142.             // and its size is its preferred size
143.             Dimension d = c.getPreferredSize();
144.             c.setBounds(x - d.width / 2, y - d.height / 2, d.width, d.height);
145.          }
146.       }
147.    }
148.
149.    private int minWidth = 0;
150.    private int minHeight = 0;
151.    private int preferredWidth = 0;
152.    private int preferredHeight = 0;
153.    private boolean sizesSet = false;
154.    private int maxComponentWidth = 0;
155.    private int maxComponentHeight = 0;
156. }
```

API `java.awt.LayoutManager 1.0`

- `void addLayoutComponent(String name, Component comp)`
  将组件添加到布局中。
  参数：name　　组件位置的标识符
  　　comp　　被添加的组件
- `void removeLayoutComponent(Component comp)`
  从本布局中删除一个组件。
- `Dimension preferredLayoutSize(Container cont)`
  返回本布局下的容器的首选尺寸。
- `Dimension minimumLayoutSize(Container cont)`
  返回本布局中下容器的最小尺寸。
- `void layoutContainer(Container cont)`
  布局容器内的组件。

9.6.5 遍历顺序

当把很多组件添加到窗口中时，需要考虑遍历顺序（traversal order）的问题。窗口被初次显示时，遍历序列的第一个组件会有键盘焦点。每次用户按下TAB键，下一个组件就会获得焦点（回忆一下，具有键盘焦点的组件可以用键盘进行操作。例如，如果按钮具有焦点，按下空格键就相当于“点击”它）。我们可能并不习惯使用TAB键遍历一组控件，但是也有很多用户喜欢这样做。这些人有可能厌恶鼠标，也有可能由于残疾而无法使用鼠标，或者采用语言方式进行交互，所以应该知道Swing是如何设置遍历顺序的。

遍历顺序很直观，它的顺序是从左至右，从上至下。例如，在字体对话框例子中，组件按照下面顺序进行遍历（如图9-35）：

①外观组合框
②示例文本区（按下CTRL+TAB键移动到下一个文本域，TAB字符被认为是文本输入）。
③尺寸组合框
④加粗复选框
⑤斜体复选框

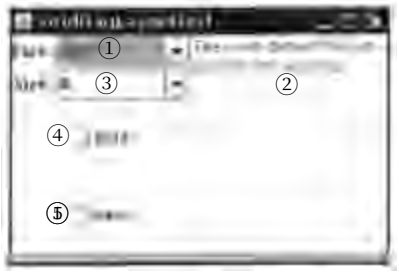


图9-35 地理位置遍历顺序


注释：在早期的AWT中，遍历顺序是由组件插入到容器中的顺序决定。在Swing中，遍历顺序仅取决于组件的布局，而不是插入顺序。

如果容器还包含其他的容器，情况就更加复杂了。当焦点给予另外一个容器时，那个容器左上角的组件就会自动地获得焦点，然后再遍历那个容器中的所有组件。最后，将焦点移交给紧跟着那个容器的组件上。

利用这一点，可以将相关元素组织在一起并放置在一个容器中。例如，放置在一个面板中。


注释：在Java SE 1.4中，调用

```
component.setFocusable(false);
```

可以从焦点遍历中删除一个组件。在早期的版本中，必须覆盖isFocusTraversable方法，但是这个方法现在已经不用了。

总之，在Java SE 1.4中有两个标准的遍历策略：

- 纯的AWT应用程序使用DefaultFocusTraversalPolicy。如果组件是可见的，可显示的、激活的、可聚焦的，并且它们的同质对等体也是可聚焦的，那么这些组件就包含在焦点遍历中。组件按照它们插入到容器的先后次序进行遍历。
- Swing应用程序使用LayoutFocusTraversalPolicy。如果组件是可见的，可显示的、激活的以及可聚焦的，那么这些组件就包含在焦点遍历中。组件按照它们的位置顺序进行遍历：从左至右，从上至下。不过，容器将会引入一个新的“循环”，其中的所有组件将会在这个容器的后继组件获得焦点之前被遍历。


注释：“循环”这个概念有点令人迷惑不解。在遍历到子容器的最后一个元素之后，焦点并不回到第一个元素上，而是跳到容器的后继组件上。API支持真正的循环，包括通过击

键在一个循环层次中上下移动。然而，标准的遍历政策不使用层次循环，而是将循环层次变为线性（深度优先）遍历。

注释：在Java SE 1.3中，可以调用JComponent类中的setNextFocusableComponent方法来改变默认的遍历顺序。现在已经不再使用这个方法了。要想改变遍历的顺序，可以将相关的组件放置在一个面板中以形成循环。如果这样不奏效，就需要安装一个比较器，对组件重新进行排序，或者完全替代遍历策略。然而，似乎哪个方法都没有解决根本问题——请参看Sun API文档。

9.7 对话框

到目前为止，所有的用户界面组件都显示在应用程序创建的框架窗口中。这对于编写运行在Web浏览器中的applets来说是十分常见的情况。但是，如果编写应用程序，通常就需要弹出独立的对话框来显示信息或者获取用户信息。

与大多数的窗口系统一样，AWT也分为模式对话框和无模式对话框。所谓模式对话框是指在结束对它的处理之前，不允许用户与应用程序的其余窗口进行交互。模式对话框主要用于在程序继续运行之前获取用户提供的信息。例如，当用户想要读取文件时，就会弹出一个模式对话框。用户必须给定一个文件名，然后程序才能够开始读操作。只有用户关闭（模式）对话框之后，应用程序才能够继续执行。

所谓无模式对话框是指允许用户同时在对话框和应用程序的其他窗口中输入信息。使用无模式对话框的最好例子就是工具栏。工具栏可以停靠在任何地方，并且用户可以在需要的时候，同时与应用程序窗口和工具栏进行交互。

本节从最简单的对话框开始——一个简单信息的模式对话框。Swing有一个很容易使用的类JOptionPane，它可以弹出一个简单的对话框，而不必编写任何对话框的相关代码。随后，将看到如何通过实现自己的对话框窗口来编写一个复杂的对话框。最后，介绍在应用程序与对话框之间如何传递数据。

本节用两个标准的对话框结束：文件对话框和颜色对话框。文件对话框比较复杂，为此需要熟悉Swing中的JFileChooser——自己编写文件对话框是一项颇有挑战性的任务。JColorChooser对话框可用来让用户选取颜色。

9.7.1 选项对话框

Swing有一套简单的对话框，用于获取用户的一些简单信息。JOptionPane有4个用于显示这些对话框的静态方法：

`showMessageDialog`：	显示一条消息并等待用户点击OK
`showConfirmDialog`：	显示一条消息并等待用户确认（与OK/Cancel类似）
`showOptionDialog`：	显示一条消息并获得用户在一组选项中的选择
`showInputDialog`：	显示一条消息并获得用户输入的一行文本

图9-36 选项对话框

图9-36显示了一个典型的对话框。可以看到，对话框有下列组件：

• 一个图标
• 一条消息
• 一个或多个按钮

输入对话框有一个用于接收用户输入的额外组件。它既可能是用于输入任何字符串的文本域，也可能是允许用户从中选择的组合框。

这些对话框的确切布局和为标准消息类型选择的图标都取决于具体的观感。

左侧的图标将由下面5种消息类型决定：

```
ERROR_MESSAGE
INFORMATION_MESSAGE
WARNING_MESSAGE
QUESTION_MESSAGE
PLAIN_MESSAGE
```

PLAIN_MESSAGE类型没有图标。每个对话框类型都有一个方法，可以用来提供自己的图标，以替代原来的图标。

可以为每个对话框类型指定一条消息。这里的消息既可以是字符串、图标、用户界面组件，也可以是其他类型的对象。下面是显示消息对象的基本方式：

String：	绘制字符串
Icon：	显示图标
Component：	显示组件
Object[]：	显示数组中的所有对象，依次叠加
any other object：	调用toString方法来显示结果字符串

可以运行例9-14程序，查看一下这些选项。

当然，提供字符串消息是最常见的情况，而提供一个Component会带来更大的灵活性。这是因为通过调用paintComponent方法可以绘制自己想要的任何内容。

位于底部的按钮取决于对话框类型和选项类型。当调用showMessageDialog 和showInputDialog时，只能看到一组标准按钮（分别是OK/Cancel）。当调用showConfirmDialog时，可以选择下面四种选项类型之一：

```
DEFAULT_OPTION
YES_NO_OPTION
YES_NO_CANCEL_OPTION
OK_CANCEL_OPTION
```

使用showOptionDialog 可以指定任意的选项。这里需要为选项提供一个对象数组。每个数组元素可以是下列类型之一：

String：	使用字符串标签创建一个按钮
Icon：	使用图标创建一个按钮
Component：	显示这个组件

其他类型的对象：使用toString方法，然后用结果字符串作为标签创建按钮

下面是这些方法的返回值：

showMessageDialog	无
showConfirmDialog	表示被选项的一个整数

showOptionDialog	表示被选项的一个整数
showInputDialog	用户选择或输入的字符串

showConfirmDialog和showOptionDialog返回一个整数用来表示用户选择了哪个按钮。对于选项对话框来说，这个值就是被选的选项的索引值或者是CLOSED_OPTION（此时用户没有选择可选项，而是关闭了对话框）。对于确认对话框，返回值可以是下列值之一：

```
OK_OPTION
CANCEL_OPTION
YES_OPTION
NO_OPTION
CLOSED_OPTION
```

这些选项似乎令人感到迷惑不解，实际上非常简单：

1）选择对话框的类型（消息、确认、选项或者输入）。

2）选择图标（错误、信息、警告、问题、无或者自定义）。

3）选择消息（字符串、图表、自定义组件或者它们的集合）。

4）对于确认对话框，选择选项类型（默认、Yes/No、Yes/No/Cancel或者Ok/Cancel）。

5）对于选项对话框，选择选项（字符串、图表或者自定义组件）和默认选项。

6）对于输入对话框，选择文本框或者组合框。

7）调用JOptionPane API中的相应方法。

例如，假设需要显示一个图9-36的对话框。这个对话框显示了一条消息，并请求用户确认或者取消。这是一个确认对话框。图标是警告图标，消息是字符串，选项类型是OK_CANCEL_OPTION。调用如下：

```
int selection = JOptionPane.showConfirmDialog(parent,
   "Message", "Title",
   JOptionPane.OK_CANCEL_OPTION,
   JOptionPane.QUESTION_MESSAGE);
if (selection == JOptionPane.OK_OPTION) . . .
```

提示：消息字符串中可以包含换行符（'\n'）。这样就可以让字符串显示在多行。

例9-14中的程序可以让用户进行选择，如图9-37所示，然后显示结果对话框。

例9-14 OptionDialogTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.awt.geom.*;
4. import java.util.*;
5. import javax.swing.*;
6.
7. /**
8.  * @version 1.33 2007-04-28
9.  * @author Cay Horstmann
10. */
11. public class OptionDialogTest
12. {
13.    public static void main(String[] args)
```

```
14.    {
15.       EventQueue.invokeLater(new Runnable()
16.          {
17.             public void run()
18.             {
19.                OptionDialogFrame frame = new OptionDialogFrame();
20.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21.                frame.setVisible(true);
22.             }
23.          });
24.    }
25. }
26.
27. /**
28.  * A panel with radio buttons inside a titled border.
29.  */
30. class ButtonPanel extends JPanel
31. {
32.    /**
33.     * Constructs a button panel.
34.     * @param title the title shown in the border
35.     * @param options an array of radio button labels
36.     */
37.    public ButtonPanel(String title, String... options)
38.    {
39.       setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(), title));
40.       setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
41.       group = new ButtonGroup();
42.
43.       // make one radio button for each option
44.       for (String option : options)
45.       {
46.          JRadioButton b = new JRadioButton(option);
47.          b.setActionCommand(option);
48.          add(b);
49.          group.add(b);
50.          b.setSelected(option == options[0]);
51.       }
52.    }
53.
54.    /**
55.     * Gets the currently selected option.
56.     * @return the label of the currently selected radio button.
57.     */
58.    public String getSelection()
59.    {
60.       return group.getSelection().getActionCommand();
61.    }
62.
63.    private ButtonGroup group;
64. }
65.
66. /**
67.  * A frame that contains settings for selecting various option dialogs.
68.  */
69. class OptionDialogFrame extends JFrame
70. {
```

```
71.    public OptionDialogFrame()
72.    {
73.       setTitle("OptionDialogTest");
74.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
75.
76.       JPanel gridPanel = new JPanel();
77.       gridPanel.setLayout(new GridLayout(2, 3));
78.
79.       typePanel = new ButtonPanel("Type", "Message", "Confirm", "Option", "Input");
80.       messageTypePanel = new ButtonPanel("Message Type", "ERROR_MESSAGE", "INFORMATION_MESSAGE",
81.             "WARNING_MESSAGE", "QUESTION_MESSAGE", "PLAIN_MESSAGE");
82.       messagePanel = new ButtonPanel("Message", "String", "Icon", "Component", "Other", "Object[]");
83.       optionTypePanel = new ButtonPanel("Confirm", "DEFAULT_OPTION", "YES_NO_OPTION",
84.             "YES_NO_CANCEL_OPTION", "OK_CANCEL_OPTION");
85.       optionsPanel = new ButtonPanel("Option", "String[]", "Icon[]", "Object[]");
86.       inputPanel = new ButtonPanel("Input", "Text field", "Combo box");
87.
88.       gridPanel.add(typePanel);
89.       gridPanel.add(messageTypePanel);
90.       gridPanel.add(messagePanel);
91.       gridPanel.add(optionTypePanel);
92.       gridPanel.add(optionsPanel);
93.       gridPanel.add(inputPanel);
94.
95.       // add a panel with a Show button
96.
97.       JPanel showPanel = new JPanel();
98.       JButton showButton = new JButton("Show");
99.       showButton.addActionListener(new ShowAction());
100.      showPanel.add(showButton);
101.
102.      add(gridPanel, BorderLayout.CENTER);
103.      add(showPanel, BorderLayout.SOUTH);
104.   }
105.
106.   /**
107.    * Gets the currently selected message.
108.    * @return a string, icon, component or object array, depending on the Message panel selection
109.    */
110.   public Object getMessage()
111.   {
112.      String s = messagePanel.getSelection();
113.      if (s.equals("String")) return messageString;
114.      else if (s.equals("Icon")) return messageIcon;
115.      else if (s.equals("Component")) return messageComponent;
116.      else if (s.equals("Object[]")) return new Object[] { messageString, messageIcon,
117.            messageComponent, messageObject };
118.      else if (s.equals("Other")) return messageObject;
119.      else return null;
120.   }
121.
122.   /**
123.    * Gets the currently selected options.
124.    * @return an array of strings, icons or objects, depending on the Option panel selection
125.    */
126.   public Object[] getOptions()
127.   {
```

```
128.    String s = optionsPanel.getSelection();
129.    if (s.equals("String[]")) return new String[] { "Yellow", "Blue", "Red" };
130.    else if (s.equals("Icon[]")) return new Icon[] { new ImageIcon("yellow-ball.gif"),
131.          new ImageIcon("blue-ball.gif"), new ImageIcon("red-ball.gif") };
132.    else if (s.equals("Object[]")) return new Object[] { messageString, messageIcon,
133.          messageComponent, messageObject };
134.    else return null;
135. }
136.
137. /**
138.  * Gets the selected message or option type
139.  * @param panel the Message Type or Confirm panel
140.  * @return the selected XXX_MESSAGE or XXX_OPTION constant from the JOptionPane class
141.  */
142. public int getType(ButtonPanel panel)
143. {
144.    String s = panel.getSelection();
145.    try
146.    {
147.       return JOptionPane.class.getField(s).getInt(null);
148.    }
149.    catch (Exception e)
150.    {
151.       return -1;
152.    }
153. }
154.
155. /**
156.  * The action listener for the Show button shows a Confirm, Input, Message or Option dialog
157.  * depending on the Type panel selection.
158.  */
159. private class ShowAction implements ActionListener
160. {
161.    public void actionPerformed(ActionEvent event)
162.    {
163.       if (typePanel.getSelection().equals("Confirm")) JOptionPane.showConfirmDialog(
164.             OptionDialogFrame.this, getMessage(), "Title", getType(optionTypePanel),
165.             getType(messageTypePanel));
166.       else if (typePanel.getSelection().equals("Input"))
167.       {
168.          if (inputPanel.getSelection().equals("Text field")) JOptionPane.showInputDialog(
169.                OptionDialogFrame.this, getMessage(), "Title", getType(messageTypePanel));
170.          else JOptionPane.showInputDialog(OptionDialogFrame.this, getMessage(), "Title",
171.                getType(messageTypePanel), null, new String[] { "Yellow", "Blue", "Red" },
172.                "Blue");
173.       }
174.       else if (typePanel.getSelection().equals("Message")) JOptionPane.showMessageDialog(
175.             OptionDialogFrame.this, getMessage(), "Title", getType(messageTypePanel));
176.       else if (typePanel.getSelection().equals("Option")) JOptionPane.showOptionDialog(
177.             OptionDialogFrame.this, getMessage(), "Title", getType(optionTypePanel),
178.             getType(messageTypePanel), null, getOptions(), getOptions()[0]);
179.    }
180. }
181.
182. public static final int DEFAULT_WIDTH = 600;
183. public static final int DEFAULT_HEIGHT = 400;
```

```
184.
185.    private ButtonPanel typePanel;
186.    private ButtonPanel messagePanel;
187.    private ButtonPanel messageTypePanel;
188.    private ButtonPanel optionTypePanel;
189.    private ButtonPanel optionsPanel;
190.    private ButtonPanel inputPanel;
191.
192.    private String messageString = "Message";
193.    private Icon messageIcon = new ImageIcon("blue-ball.gif");
194.    private Object messageObject = new Date();
195.    private Component messageComponent = new SampleComponent();
196. }
197.
198. /**
199.  * A component with a painted surface
200.  */
201.
202. class SampleComponent extends JComponent
203. {
204.    public void paintComponent(Graphics g)
205.    {
206.       Graphics2D g2 = (Graphics2D) g;
207.       Rectangle2D rect = new Rectangle2D.Double(0, 0, getWidth() - 1, getHeight() - 1);
208.       g2.setPaint(Color.YELLOW);
209.       g2.fill(rect);
210.       g2.setPaint(Color.BLUE);
211.       g2.draw(rect);
212.    }
213.
214.    public Dimension getPreferredSize()
215.    {
216.       return new Dimension(10, 10);
217.    }
218. }
```

图9-37　OptionDialogTest程序

API javax.swing.JOptionPane 1.2

- static void showMessageDialog(Component parent, Object message, String title, int messageType, Icon icon)
- static void showMessageDialog(Component parent, Object message, String title, int messageType)
- static void showMessageDialog(Component parent, Object message)
- static void showInternalMessageDialog(Component parent, Object message, String title, int messageType, Icon icon)
- static void showInternalMessageDialog(Component parent, Object message, String title, int messageType)
- static void showInternalMessageDialog(Component parent, Object message)

显示一个消息对话框或者一个内部消息对话框（内部对话框完全显示在所在的框架内）。

参数：parent 父组件（可以为null）。

message 显示在对话框中的消息（可以是字符串、图标、组件或者一个这些类型的数组）。

title 对话框标题栏中的字符串。

messageType 取值为 ERROR_MESSAGE、INFORMATION_MESSAGE、WARNING_MESSAGE、QUESTION_MESSAGE、PLAIN_MESSAGE之一。

icon 用于替代标准图标的图标。

- static int showConfirmDialog(Component parent, Object message, String title, int optionType, int messageType, Icon icon)
- static int showConfirmDialog(Component parent, Object message, String title, int optionType, int messageType)
- static int showConfirmDialog(Component parent, Object message, String title, int optionType)
- static int showConfirmDialog(Component parent, Object message)
- static int showInternalConfirmDialog(Component parent, Object message, String title, int optionType, int messageType, Icon icon)
- static int showInternalConfirmDialog(Component parent, Object message, String title, int optionType, int messageType)
- static int showInternalConfirmDialog(Component parent, Object message, String title, int optionType)
- static int showInternalConfirmDialog(Component parent, Object message)

显示一个确认对话框或者内部确认对话框（内部对话框完全显示在所在的框架内）。返回用户选择的选项（取值为OK_OPTION, CANCEL_OPTION, YES_OPTION, NO_OPTION）；如

果用户关闭对话框将返回CLOSED_OPTION。

参数：parent　父组件（可以为null）。
message　显示在对话框中的消息（可以是字符串、图标、组件或者一个这些类型的数组）。
title　对话框标题栏中的字符串。
messageType　取值为ERROR_MESSAGE、INFORMATION_MESSAGE、WARNING_MESSAGE、QUESTION_MESSAGE、PLAIN_MESSAGE之一。
optionType　取值为DEFAULT_OPTION、YES_NO_OPTION、YES_NO_CANCEL_OPTION、OK_CANCEL_OPTION之一。
icon　用于替代标准图标的图标。

- `static int showOptionDialog(Component parent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object default)`
- `static int showInternalOptionDialog(Component parent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object default)`

显示一个选项对话框或者内部选项对话框（内部对话框完全显示在所在的框架内）。返回用户选择的选项索引；如果用户取消对话框返回CLOSED_OPTION。

参数：parent　父组件（可以为null）。
message　显示在对话框中的消息（可以是字符串，图标，组件或者一个这些类型的数组）。
title　对话框标题栏中的字符串。
messageType　取值为ERROR_MESSAGE、INFORMATION_MESSAGE、WARNING_MESSAGE、QUESTION_MESSAGE、PLAIN_MESSAGE之一。
optionType　取值为DEFAULT_OPTION、YES_NO_OPTION、YES_NO_CANCEL_OPTION、OK_CANCEL_OPTION之一。
icon　用于替代标准图标的图标。
options　一组选项（可以是字符串、图标或者组件）。
default　呈现给用户的默认值。

- `static Object showInputDialog(Component parent, Object message, String title, int messageType, Icon icon, Object[] values, Object default)`
- `static String showInputDialog(Component parent, Object message, String title, int messageType)`
- `static String showInputDialog(Component parent, Object message)`
- `static String showInputDialog(Object message)`
- `static String showInputDialog(Component parent, Object message, Object`

default) 1.4

- static String showInputDialog(Object message, Object default) 1.4
- static Object showInternalInputDialog(Component parent, Object message, String title, int messageType, Icon icon, Object[] values, Object default)
- static String showInternalInputDialog(Component parent, Object message, String title, int messageType)
- static String showInternalInputDialog(Component parent, Object message)

显示一个输入对话框或者内部输入对话框（内部对话框完全显示在所在的框架内）。返回用户输入的字符串；如果用户取消对话框返回null。

参数：parent 父组件（可以为null）。

message 显示在对话框中的消息（可以是字符串、图标、组件或者一个这些类型的数组）。

title 对话框标题栏中的字符串。

messageType 取值为ERROR_MESSAGE、INFORMATION_MESSAGE、WARNING_MESSAGE、QUESTION_MESSAGE、PLAIN_MESSAGE之一。

icon 用于替代标准图标的图标。

values 在组合框中显示的一组值。

default 呈现给用户的默认值。

9.7.2 创建对话框

在上一节中，介绍了如何使用JOptionPane来显示一个简单的对话框。本节将讲述如何手工地创建这样一个对话框。

图9-38显示了一个典型的模式对话框。当用户点击About按钮时就会显示这样一个程序信息对话框。

图9-38 About对话框

要想创建一个对话框，需要从JDialog派生一个类。这与应用程序窗口派生于JFrame的过程完全一样。具体过程如下：

1）在对话框构造器中，调用超类JDialog的构造器。

2）添加对话框的用户界面组件。

3）添加事件处理器。

4）设置对话框的大小。

在调用超类构造器时，需要提供拥有者框架（owner frame）、对话框标题及特征。

拥有者框架控制对话框的显示位置，如果将拥有者标识为null，那么对话框将由一个隐藏框架所拥有。

特征将指定对话框处于显示状态时，应用程序中其他窗口是否被锁住。无模式对话框不会锁住其他窗口，而有模式对话框将锁住应用程序中的所有其他窗口（除对话框的子窗口外）。

用户经常使用的工具栏就是无模式对话框，换句话说，如果想强迫用户在继续操作之前提供一些必要的信息就应该使用模式对话框。

注释：在Java SE 6.中，有两个额外的特征类型。文档 - 模式对话框将阻塞所有属于相同“文档”的窗口。更准确地说，是所有作为对话框的具有相同无父根窗口的窗口。这样解决了帮助系统的问题。在早期的版本中，当弹出一个模式对话框时，用户不可能与帮助窗口结合。工具箱对话框阻塞了所有来自于相同“工具箱”的窗口。工具箱是一个运行于多个应用的Java程序，例如，浏览器中的小应用程序引擎。有关更加详细的内容请参看网站：http://java.sun.com/developer/technicalArticles/J2SE/Desktop/javase6/modality。

下面是一个对话框的例子：

```
public AboutDialog extends JDialog
{
   public AboutDialog(JFrame owner)
   {
      super(owner, "About DialogTest", true);
      add(new JLabel(
         "<html><h1><i>Core Java</i></h1><hr>By Cay Horstmann and Gary Cornell</html>"),
         BorderLayout.CENTER);

      JPanel panel = new JPanel();
      JButton ok = new JButton("Ok");

      ok.addActionListener(new
         ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               setVisible(false);
            }
         });
      panel.add(ok);
      add(panel, BorderLayout.SOUTH);

      setSize(250, 150);
   }
}
```

正如看到的，构造器添加了用户界面组件，在本例中添加是标签和按钮，并且为按钮设置了处理器，然后还设置了对话框的大小。

要想显示对话框，需要建立一个新的对话框对象，并让它可见：

```
JDialog dialog = new AboutDialog(this);
dialog.setVisible(true);
```

实际上，在下面的示例代码中，只建立了一次对话框，无论何时用户点击About按钮，都可以重复使用它。

```
if (dialog == null) // first time
   dialog = new AboutDialog(this);
dialog.setVisible(true);
```

当用户点击OK按钮时，该对话框将被关闭。下面是在OK按钮的事件处理器中的处理代码：

```
ok.addActionListener(new
   ActionListener()
   {
      public void actionPerformed(ActionEvent event)
      {
         setVisible(false);
      }
   });
```

当用户点击Close按钮关闭对话框时，对话框就被隐藏起来。与JFrame一样，可以覆盖setDefaultCloseOperation方法来改变这个行为。

例9-15 是About对话框的测试程序代码。

例9-15 DialogTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.33 2007-06-12
7.  * @author Cay Horstmann
8.  */
9. public class DialogTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                DialogFrame frame = new DialogFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame with a menu whose File->About action shows a dialog.
27.  */
28. class DialogFrame extends JFrame
29. {
30.    public DialogFrame()
31.    {
32.       setTitle("DialogTest");
33.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
34.
35.       // construct a File menu
36.
37.       JMenuBar menuBar = new JMenuBar();
38.       setJMenuBar(menuBar);
39.       JMenu fileMenu = new JMenu("File");
40.       menuBar.add(fileMenu);
41.
```

```
42.       // add About and Exit menu items
43.
44.       // The About item shows the About dialog
45.
46.       JMenuItem aboutItem = new JMenuItem("About");
47.       aboutItem.addActionListener(new ActionListener()
48.          {
49.             public void actionPerformed(ActionEvent event)
50.             {
51.                if (dialog == null) // first time
52.                dialog = new AboutDialog(DialogFrame.this);
53.                dialog.setVisible(true); // pop up dialog
54.             }
55.          });
56.       fileMenu.add(aboutItem);
57.
58.       // The Exit item exits the program
59.
60.       JMenuItem exitItem = new JMenuItem("Exit");
61.       exitItem.addActionListener(new ActionListener()
62.          {
63.             public void actionPerformed(ActionEvent event)
64.             {
65.                System.exit(0);
66.             }
67.          });
68.       fileMenu.add(exitItem);
69.    }
70.
71.    public static final int DEFAULT_WIDTH = 300;
72.    public static final int DEFAULT_HEIGHT = 200;
73.
74.    private AboutDialog dialog;
75. }
76.
77. /**
78.  * A sample modal dialog that displays a message and waits for the user to click the Ok button.
79.  */
80. class AboutDialog extends JDialog
81. {
82.    public AboutDialog(JFrame owner)
83.    {
84.       super(owner, "About DialogTest", true);
85.
86.       // add HTML label to center
87.
88.       add(
89.             new JLabel(
90.                "<html><h1><i>Core Java</i></h1><hr>By Cay Horstmann and Gary Cornell</html>"),
91.             BorderLayout.CENTER);
92.
93.       // Ok button closes the dialog
94.
95.       JButton ok = new JButton("Ok");
96.       ok.addActionListener(new ActionListener()
97.          {
98.             public void actionPerformed(ActionEvent event)
```

```
99.             {
100.               setVisible(false);
101.            }
102.         });
103.
104.      // add Ok button to southern border
105.
106.      JPanel panel = new JPanel();
107.      panel.add(ok);
108.      add(panel, BorderLayout.SOUTH);
109.
110.      setSize(250, 150);
111.   }
112. }
```

API javax.swing.JDialog 1.2

- public JDialog(Frame parent, String title, boolean modal)

  构造一个对话框。在没有明确地让对话框显示之前，它是不可见的。

  参数：parent　对话框拥有者的框架

  title　对话框的标题

  modal　True代表模式对话框（模式对话框阻隔其他窗口的输入）

9.7.3 数据交换

使用对话框最通常的目的是获取用户的输入信息。在前面已经看到，构造对话框对象非常简单：首先初始化数据，然后调用setVisible(true)就会在屏幕上显示对话框。现在，看看如何将数据传入传出对话框。

图9-39 密码对话框

看一下如图9-39所示的对话框，可以用来获得用户名和用户密码以便连接某些在线服务。

对话框应该提供设置默认数据的方法。例如，示例程序中的PasswordChooser类提供了一个setUser方法，用来将默认值放到下面的字段中：

```
public void setUser(User u)
{
   username.setText(u.getName());
}
```

一旦设置了默认值（如果需要），就可以调用setVisible(true)让对话框显示在屏幕上。

然后用户输入信息，点击OK或者Cancel按钮。这两个按钮的事件处理器都会调用setVisible(false)终止对setVisible(true)的调用。另外，用户也可以选择关闭对话框。如果没有为对话框安装窗口监听器，就会执行默认的窗口结束操作，即对话框变为不可见，这也中止了对setVisible(true)的调用。

重要的问题是在用户解除这个对话框之前，一直调用setVisible(true)阻塞。这样易于实现

模式对话框。

希望知道用户是接收对话框，还是取消对话框。在示例代码中设置了OK标志，在对话框显示之前是false。只有OK按钮的事件处理器可以将它设置为true。这样，就可以获得对话框中的用户输入。

注释：无模式对话框数据传输就没有那么简单了。当无模式对话框显示时，调用setVisible(true)并不阻塞，在对话框显示时，其他程序仍继续运行。如果用户选择了无模式对话框中的一项，并点击OK，对话框就会将一个事件发送给程序中的某个监听器。

示例程序中还包含另外一个很有用的改进。在构造一个JDialog对象时，需要指定拥有者框架。但是，在很多情况下，一个对话框可能会有多个拥有者框架，所以最好在准备显示对话框时再确定拥有者框架，而不是在构造PasswordChooser对象时。

有一个技巧是让PasswordChooser扩展于JPanel，而不是扩展于JDialog，在showDialog方法中立即建立JDialog对象：

```
public boolean showDialog(Frame owner, String title)
{
   ok = false;

   if (dialog == null || dialog.getOwner() != owner)
   {
      dialog = new JDialog(owner, true);
      dialog.add(this);
      dialog.pack();
   }

   dialog.setTitle(title);
   dialog.setVisible(true);
   return ok;
}
```

注意，让owner等于null比较安全。

可以再做进一步的改进。有时，拥有者框架并不总是可用的。利用任意的parent组件可以很容易地得到它。如下所示：

```
Frame owner;
if (parent instanceof Frame)
   owner = (Frame) parent;
else
   owner = (Frame) SwingUtilities.getAncestorOfClass(Frame.class, parent);
```

在示例程序中使用了改进的代码。JOptionPane类也使用了上面的机制。

很多对话框都有默认按钮。如果用户按下一个触发器键（在大多数“观感”实现中是ENTER）就自动地选择了它。默认按钮通常用加粗的轮廓给予特别的标识。

可以在对话框的根窗格（root pane）中设置默认按钮：

```
dialog.getRootPane().setDefaultButton(okButton);
```

如果按照前面的建议，在一个面板中布局对话框就必须特别小心。在包装面板进入对话框后再设置默认按钮。面板本身没有根窗格。

例9-16是一个描述对话框输入和输出数据流的完整程序。

例9-16 DataExchangeTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4.
5. /**
6.  * @version 1.33 2007-06-12
7.  * @author Cay Horstmann
8.  */
9. public class DataExchangeTest
10. {
11.    public static void main(String[] args)
12.    {
13.       EventQueue.invokeLater(new Runnable()
14.          {
15.             public void run()
16.             {
17.                DataExchangeFrame frame = new DataExchangeFrame();
18.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.                frame.setVisible(true);
20.             }
21.          });
22.    }
23. }
24.
25. /**
26.  * A frame with a menu whose File->Connect action shows a password dialog.
27.  */
28. class DataExchangeFrame extends JFrame
29. {
30.    public DataExchangeFrame()
31.    {
32.       setTitle("DataExchangeTest");
33.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
34.
35.       // construct a File menu
36.
37.       JMenuBar mbar = new JMenuBar();
38.       setJMenuBar(mbar);
39.       JMenu fileMenu = new JMenu("File");
40.       mbar.add(fileMenu);
41.
42.       // add Connect and Exit menu items
43.
44.       JMenuItem connectItem = new JMenuItem("Connect");
45.       connectItem.addActionListener(new ConnectAction());
46.       fileMenu.add(connectItem);
47.
48.       // The Exit item exits the program
49.
50.       JMenuItem exitItem = new JMenuItem("Exit");
51.       exitItem.addActionListener(new ActionListener()
52.          {
53.             public void actionPerformed(ActionEvent event)
```

```
54.             {
55.                System.exit(0);
56.             }
57.          });
58.       fileMenu.add(exitItem);
59.
60.       textArea = new JTextArea();
61.       add(new JScrollPane(textArea), BorderLayout.CENTER);
62.    }
63.
64.    public static final int DEFAULT_WIDTH = 300;
65.    public static final int DEFAULT_HEIGHT = 200;
66.
67.    private PasswordChooser dialog = null;
68.    private JTextArea textArea;
69.
70.    /**
71.     * The Connect action pops up the password dialog.
72.     */
73.
74.    private class ConnectAction implements ActionListener
75.    {
76.       public void actionPerformed(ActionEvent event)
77.       {
78.          // if first time, construct dialog
79.
80.          if (dialog == null) dialog = new PasswordChooser();
81.
82.          // set default values
83.          dialog.setUser(new User("yourname", null));
84.
85.          // pop up dialog
86.          if (dialog.showDialog(DataExchangeFrame.this, "Connect"))
87.          {
88.             // if accepted, retrieve user input
89.             User u = dialog.getUser();
90.             textArea.append("user name = " + u.getName() + ", password = "
91.                   + (new String(u.getPassword())) + "\n");
92.          }
93.       }
94.    }
95. }
96.
97. /**
98.  * A password chooser that is shown inside a dialog
99.  */
100. class PasswordChooser extends JPanel
101. {
102.    public PasswordChooser()
103.    {
104.       setLayout(new BorderLayout());
105.
106.       // construct a panel with user name and password fields
107.
108.       JPanel panel = new JPanel();
109.       panel.setLayout(new GridLayout(2, 2));
110.       panel.add(new JLabel("User name:"));
```

```
111.       panel.add(username = new JTextField(""));
112.       panel.add(new JLabel("Password:"));
113.       panel.add(password = new JPasswordField(""));
114.       add(panel, BorderLayout.CENTER);
115.
116.       // create Ok and Cancel buttons that terminate the dialog
117.
118.       okButton = new JButton("Ok");
119.       okButton.addActionListener(new ActionListener()
120.          {
121.             public void actionPerformed(ActionEvent event)
122.             {
123.                ok = true;
124.                dialog.setVisible(false);
125.             }
126.          });
127.
128.       JButton cancelButton = new JButton("Cancel");
129.       cancelButton.addActionListener(new ActionListener()
130.          {
131.             public void actionPerformed(ActionEvent event)
132.             {
133.                dialog.setVisible(false);
134.             }
135.          });
136.
137.       // add buttons to southern border
138.
139.       JPanel buttonPanel = new JPanel();
140.       buttonPanel.add(okButton);
141.       buttonPanel.add(cancelButton);
142.       add(buttonPanel, BorderLayout.SOUTH);
143.    }
144.
145.    /**
146.     * Sets the dialog defaults.
147.     * @param u the default user information
148.     */
149.    public void setUser(User u)
150.    {
151.       username.setText(u.getName());
152.    }
153.
154.    /**
155.     * Gets the dialog entries.
156.     * @return a User object whose state represents the dialog entries
157.     */
158.    public User getUser()
159.    {
160.       return new User(username.getText(), password.getPassword());
161.    }
162.
163.    /**
164.     * Show the chooser panel in a dialog
165.     * @param parent a component in the owner frame or null
166.     * @param title the dialog window title
167.     */
```

```
168.    public boolean showDialog(Component parent, String title)
169.    {
170.       ok = false;
171.
172.       // locate the owner frame
173.
174.       Frame owner = null;
175.       if (parent instanceof Frame) owner = (Frame) parent;
176.       else owner = (Frame) SwingUtilities.getAncestorOfClass(Frame.class, parent);
177.
178.       // if first time, or if owner has changed, make new dialog
179.
180.       if (dialog == null || dialog.getOwner() != owner)
181.       {
182.          dialog = new JDialog(owner, true);
183.          dialog.add(this);
184.          dialog.getRootPane().setDefaultButton(okButton);
185.          dialog.pack();
186.       }
187.
188.       // set title and show dialog
189.
190.       dialog.setTitle(title);
191.       dialog.setVisible(true);
192.       return ok;
193.    }
194.
195.    private JTextField username;
196.    private JPasswordField password;
197.    private JButton okButton;
198.    private boolean ok;
199.    private JDialog dialog;
200. }
201.
202. /**
203.  * A user has a name and password. For security reasons, the password is stored as a char[],
204.  * not a String.
205.  */
206. class User
207. {
208.    public User(String aName, char[] aPassword)
209.    {
210.       name = aName;
211.       password = aPassword;
212.    }
213.
214.    public String getName()
215.    {
216.       return name;
217.    }
218.
219.    public char[] getPassword()
220.    {
221.       return password;
222.    }
223.
224.    public void setName(String aName)
```

```
225.    {
226.       name = aName;
227.    }
228.
229.    public void setPassword(char[] aPassword)
230.    {
231.       password = aPassword;
232.    }
233.
234.    private String name;
235.    private char[] password;
236. }
```

API javax.swing.SwingUtilities 1.2

- `Container getAncestorOfClass(Class c, Component comp)`
  返回给定组件的最先的父容器。这个组件属于给定的类或者其子类之一。

API javax.swing.JComponent 1.2

- `JRootPane getRootPane()`
  获得最靠近这个组件的根窗格，如果这个组件的祖先没有根窗格返回null。

API javax.swing.JRootPane 1.2

- `void setDefaultButton(JButton button)`
  设置根窗格的默认按钮。要想禁用默认按钮，需要将参数设置为null。

API javax.swing.JButton 1.2

- `boolean isDefaultButton()`
  如果这个按钮是它的根窗格的默认按钮，返回true。

9.7.4 文件对话框

当编写应用程序时，通常希望可以打开和保存文件。一个好的文件对话框应该可以显示文件和目录，可以让用户浏览文件系统，这是很难编写的。人们肯定不愿意从头做起。很幸运，Swing中提供了JFileChooser类，它可以显示一个文件对话框，其外观与本地应用程序中使用的文件对话框基本一样。JFileChooser是一个模式对话框。注意，JFileChooser类并不是JDialog类的子类。需要调用showOpenDialog，而不是调用setVisible(true)显示打开文件的对话框，或者调用showSaveDialog显示保存文件的对话框。接收文件的按钮被自动地标签为Open或者Save。也可以调用showDialog方法为按钮设定标签。图9-40是文件选择对话框的样例。

下面是建立文件对话框并且获取用户选择信息的步骤：

1）建立一个JFileChooser对象。与JDialog类的构造器不同，它不需要指定父组件。允许在多个框架中重用一个文件选择器。例如：

```
JFileChooser chooser = new JFileChooser();
```

图9-40 文件选择对话框

提示：重用一个文件选择器对象是一个很好的想法，其原因是JFileChooser的构造器相当耗费时间。特别是在Windows上，用户映射了很多网络驱动器的情况下。

2）调用setCurrentDirectory方法设置当前目录。例如，使用当前的工作目录：

```
chooser.setCurrentDirectory(new File("."));
```

需要提供一个File对象。File对象将在第12章中详细地介绍。这里只需要知道构造器File（String fileName）能够将一个文件或目录名转化为一个File对象即可。

3）如果有一个想要作为用户选择的默认文件名，可以使用setSelectedFile方法进行指定：

```
chooser.setSelectedFile(new File(filename));
```

4）如果允许用户在对话框中选择多个文件，需要调用setMultiSelectionEnabled方法。当然，这是可选的。

```
chooser.setMultiSelectionEnabled(true);
```

5）如果想让对话框仅显示某一种类型的文件（如，所有扩展名为.gif的文件），需要设置文件过滤器，稍后将会进行讨论。

6）在默认情况下，用户在文件选择器中只能选择文件。如果希望选择目录，需要调用setFileSelectionMode方法。参数值为：JFileChooser.FILES_ONLY（默认值），JFileChooser.DIRECTORIES_ONLY或者JFileChooser.FILES_AND_DIRECTORIES。

7）调用showOpenDialog 或者showSaveDialog方法显示对话框。必须为这些调用提供父组件：

```
int result = chooser.showOpenDialog(parent);
```

或者

```
int result = chooser.showSaveDialog(parent);
```

这些调用的区别是“确认按钮”的标签不同。点击“确认按钮”将完成文件选择。也可以调用showDialog方法，并将一个显式的文本传递给确认按钮：

```
int result = chooser.showDialog(parent, "Select");
```

仅当用户确认、取消或者离开对话框时才返回调用。返回值可以是JFileChooser. APPROVE_OPTION、JFileChooser.CANCEL_OPTION或者JFileChooser.ERROR_OPTION。

8）调用getSelectedFile() 或者 getSelectedFiles() 方法获取用户选择的一个或多个文件。这些方法将返回一个文件对象或者一组文件对象。如果需要知道文件对象名时，可以调用getPath方法。例如：

```
String filename = chooser.getSelectedFile().getPath();
```

在大多数情况下，这些过程比较简单。使用文件对话框的主要困难在于指定用户需要选择的文件子集。例如，假定用户应该选择GIF图像文件。后面的文件选择器就应该只显示扩展名为.gif的文件，并且，还应该为用户提供反馈信息来说明显示的特定文件类别，如“GIF图像”。然而，情况有可能会更加复杂。如果用户应该选择JPFG图像文件，扩展名就可以是.jpg或者.jpeg。与重新编码实现这种复杂情况相比，文件选择器的设计者提供了一种更好的机制：若想限制显示的文件，需要创建一个实现了抽象类javax.swing.filechooser.FileFilter的对象。文件选择器将每个文件传递给文件过滤器，只有文件过滤器接受的文件才被最终显示出来。

在编写本书的时候，有两个子类可用：可以接受所有文件的默认过滤器和可以接受给定扩展名的所有文件的过滤器。其实，设计专用文件过滤器非常简单，只要实现FileFilter超类中的两个方法即可：

```
public boolean accept(File f);
public String getDescription();
```

第一个方法检测是否应该接受一个文件，第二个方法返回显示在文件选择器对话框种显示的文件类型的描述信息。

注释：在java.io包中有一个无关的FileFilter接口，其中只包含一个方法：boolean accept(File f)。File类中的listFiles方法利用它显示目录中的文件。我们不知道Swing的设计者为什么不扩展这个接口，可能是因为Java类库过于复杂，致使Sun程序员也不太熟悉所有的标准类和接口了。

需要解决同时导入javax.io包和javax.swing.filechooser包带来的名称冲突问题。最简单的方法是导入javax.swing.filechooser.Filefilter，而不要导入javax.swing.filechooser.*

一旦有了文件过滤器对象，就可以调用JFileChooser类中的setFileFilter方法，将这个对象安装到文件选择器对象中：

```
chooser.setFileFilter(new FileNameExtensionFilter("Image files", "gif", "jpg");
```

可以为一个文件选择器安装多个过滤器：

```
chooser.addChoosableFileFilter(filter1);
chooser.addChoosableFileFilter(filter2);
. . .
```

用户可以从文件对话框底部的组合框中选择过滤器。在默认情况下，All files过滤器总是显示在组合框中。这是一个很好的主意，特别是在使用这个程序的用户需要选择一个具有非标准扩展名的文件时。然而，如果你想放弃All files过滤器，需要调用：

```
chooser.setAcceptAllFileFilterUsed(false)
```

警告：如果为加载和保存不同类型的文件重用一个文件选择器，就需要调用：

```
chooser.resetChoosableFilters()
```

这样可以在添加新文件过滤器之前清除旧文件过滤器。

最后，可以通过为文件选择器显示的每个文件提供特定的图标和文件描述来定制文件选择器。这需要应用一个扩展于javax.swing.filechooser包中的FileView类的对象。这是一个高级技巧。在通常情况下，不需要提供文件视图 —— 可插观感会提供。然而，如果想让某种特定的文件类型显示不同的图标，就需要安装自己的文件视图。这要扩展FileView并实现下面5五个方法：

```
Icon getIcon(File f);
String getName(File f);
String getDescription(File f);
String getTypeDescription(File f);
Boolean isTraversable(File f);
```

然后，调用setFileView方法将文件视图安装到文件选择器中。

文件选择器为每个希望显示的文件或目录调用这些方法。如果方法返回的图标、名字或描述信息为null，那么文件选择器将会构造当前观感的默认文件视图。这样处理很好，其原因是这样只需处理具有不同显示的文件类型。

文件选择器调用isTraversable方法来决定是否在用户点击一个目录的时候打开这个目录。请注意，这个方法返回一个Boolean对象，而不是boolean值。看起来似乎有点怪，但实际上很方便 —— 如果需要使用默认的视图，则返回null。文件选择器将会使用默认的文件视图。换句话说，这个方法返回的Boolean对象能给出下面三种选择：真(Boolean.TRUE)，假(Boolean.FALSE)和不关心 (null)。

在示例中包含了一个简单的文件视图类。当文件匹配文件过滤器时，这个类将会显示一个特定的图标。可以利用这个类为所有的图像文件显示一个调色图标。

```
class FileIconView extends FileView
{
   public FileIconView(FileFilter aFilter, Icon anIcon)
   {
      filter = aFilter;
      icon = anIcon;
   }

   public Icon getIcon(File f)
   {
      if (!f.isDirectory() && filter.accept(f))
         return icon;
      else return null;
   }
   private FileFilter filter;
   private Icon icon;
}
```

可以调用setFileView方法将这个文件视图安装到文件选择器中：

```
chooser.setFileView(new FileIconView(filter,
```

```
        new ImageIcon("palette.gif")));
```

文件选择器会在通过filter的所有文件旁边显示调色板图标，并且使用默认的文件视图来显示所有其他的文件。很自然，我们可以使用与文件选择器设定的一样的过滤器。

提示：可以在JDK的demo/jfc/FileChooserDemo目录下找到更实用的ExampleFileView类。它可以将图标和描述信息与所有扩展名关联起来。

最后，可以通过添加一个附件组件来定制文件对话框。例如，图9-41在文件列表旁边显示了一个预览附件。这个附件显示了当前选择文件的预览信息。

图9-41 带预览附件的文件对话框

附件可以是任何Swing组件。在这个示例中，扩展JLabel类，并将图标设置为所选的图像文件的压缩拷贝。

```
class ImagePreviewer extends JLabel
{
   public ImagePreviewer(JFileChooser chooser)
   {
      setPreferredSize(new Dimension(100, 100));
      setBorder(BorderFactory.createEtchedBorder());
   }

   public void loadImage(File f)
   {
      ImageIcon icon = new ImageIcon(f.getPath());
      if(icon.getIconWidth() > getWidth())
         icon = new ImageIcon(icon.getImage().getScaledInstance(
            getWidth(), -1, Image.SCALE_DEFAULT));
      setIcon(icon);
      repaint();
   }
}
```

这里还有一个挑战，即需要在用户选择不同的文件时更新预览图像。文件选择器使用了Java Beans机制。当它的属性发生变化时，文件选择器就会通知相关的监听器。被选择文件是

一个属性，可以通过安装PropertyChangeListener监听它。本书将在卷II中讨论这个机制。下面这段代码可以用来捕捉通知：

```
chooser.addPropertyChangeListener(new
   PropertyChangeListener()
   {
      public void propertyChange(PropertyChangeEvent event)
      {
         if (event.getPropertyName() == JFileChooser.SELECTED_FILE_CHANGED_PROPERTY)
         {
            File newFile = (File) event.getNewValue()
             // update the accessory
            . . .
         }
    }
});
```

在这个示例中，将这段代码添加到ImagePreviewer构造器中。

例9-17对第2章中的ImageViewer程序做了一定的修改。通过自定义的文件视图和预览附件文件增强了文件选择器的功能。

例9-17 FileChooserTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.beans.*;
4. import java.util.*;
5. import java.io.*;
6. import javax.swing.*;
7. import javax.swing.filechooser.*;
8. import javax.swing.filechooser.FileFilter;
9.
10. /**
11.  * @version 1.23 2007-06-12
12.  * @author Cay Horstmann
13.  */
14. public class FileChooserTest
15. {
16.    public static void main(String[] args)
17.    {
18.       EventQueue.invokeLater(new Runnable()
19.          {
20.             public void run()
21.             {
22.                ImageViewerFrame frame = new ImageViewerFrame();
23.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24.                frame.setVisible(true);
25.             }
26.          });
27.    }
28. }
29.
30. /**
31.  * A frame that has a menu for loading an image and a display area for the loaded image.
32.  */
```

```
33. class ImageViewerFrame extends JFrame
34. {
35.    public ImageViewerFrame()
36.    {
37.       setTitle("FileChooserTest");
38.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
39.
40.       // set up menu bar
41.       JMenuBar menuBar = new JMenuBar();
42.       setJMenuBar(menuBar);
43.
44.       JMenu menu = new JMenu("File");
45.       menuBar.add(menu);
46.
47.       JMenuItem openItem = new JMenuItem("Open");
48.       menu.add(openItem);
49.       openItem.addActionListener(new FileOpenListener());
50.
51.       JMenuItem exitItem = new JMenuItem("Exit");
52.       menu.add(exitItem);
53.       exitItem.addActionListener(new ActionListener()
54.          {
55.             public void actionPerformed(ActionEvent event)
56.             {
57.                System.exit(0);
58.             }
59.          });
60.
61.       // use a label to display the images
62.       label = new JLabel();
63.       add(label);
64.
65.       // set up file chooser
66.       chooser = new JFileChooser();
67.
68.       // accept all image files ending with .jpg, .jpeg, .gif
69.       /*
70.       final ExtensionFileFilter filter = new ExtensionFileFilter();
71.       filter.addExtension("jpg");
72.       filter.addExtension("jpeg");
73.       filter.addExtension("gif");
74.       filter.setDescription("Image files");
75.       */
76.       FileNameExtensionFilter filter = new FileNameExtensionFilter(
77.             "Image files", "jpg", "jpeg", "gif");
78.       chooser.setFileFilter(filter);
79.
80.       chooser.setAccessory(new ImagePreviewer(chooser));
81.
82.       chooser.setFileView(new FileIconView(filter, new ImageIcon("palette.gif")));
83.    }
84.
85.    /**
86.     * This is the listener for the File->Open menu item.
87.     */
88.    private class FileOpenListener implements ActionListener
89.    {
```

```
90.       public void actionPerformed(ActionEvent event)
91.       {
92.          chooser.setCurrentDirectory(new File("."));
93.
94.          // show file chooser dialog
95.          int result = chooser.showOpenDialog(ImageViewerFrame.this);
96.
97.          // if image file accepted, set it as icon of the label
98.          if (result == JFileChooser.APPROVE_OPTION)
99.          {
100.            String name = chooser.getSelectedFile().getPath();
101.            label.setIcon(new ImageIcon(name));
102.         }
103.      }
104.   }
105.
106.   public static final int DEFAULT_WIDTH = 300;
107.   public static final int DEFAULT_HEIGHT = 400;
108.
109.   private JLabel label;
110.   private JFileChooser chooser;
111. }
112.
113. /**
114.  * A file view that displays an icon for all files that match a file filter.
115.  */
116. class FileIconView extends FileView
117. {
118.    /**
119.     * Constructs a FileIconView.
120.     * @param aFilter a file filter--all files that this filter accepts will be shown with the
121.     * icon. @param anIcon--the icon shown with all accepted files.
122.     */
123.    public FileIconView(FileFilter aFilter, Icon anIcon)
124.    {
125.       filter = aFilter;
126.       icon = anIcon;
127.    }
128.
129.    public Icon getIcon(File f)
130.    {
131.       if (!f.isDirectory() && filter.accept(f)) return icon;
132.       else return null;
133.    }
134.
135.    private FileFilter filter;
136.    private Icon icon;
137. }
138.
139. /**
140.  * A file chooser accessory that previews images.
141.  */
142. class ImagePreviewer extends JLabel
143. {
144.    /**
145.     * Constructs an ImagePreviewer.
```

```
146.     * @param chooser the file chooser whose property changes trigger an image change in this
147.     * previewer
148.     */
149.    public ImagePreviewer(JFileChooser chooser)
150.    {
151.       setPreferredSize(new Dimension(100, 100));
152.       setBorder(BorderFactory.createEtchedBorder());
153.
154.       chooser.addPropertyChangeListener(new PropertyChangeListener()
155.          {
156.             public void propertyChange(PropertyChangeEvent event)
157.             {
158.                if (event.getPropertyName() == JFileChooser.SELECTED_FILE_CHANGED_PROPERTY)
159.                {
160.                   // the user has selected a new file
161.                   File f = (File) event.getNewValue();
162.                   if (f == null)
163.                   {
164.                      setIcon(null);
165.                      return;
166.                   }
167.
168.                   // read the image into an icon
169.                   ImageIcon icon = new ImageIcon(f.getPath());
170.
171.                   // if the icon is too large to fit, scale it
172.                   if (icon.getIconWidth() > getWidth()) icon = new ImageIcon(icon.getImage()
173.                         .getScaledInstance(getWidth(), -1, Image.SCALE_DEFAULT));
174.
175.                   setIcon(icon);
176.                }
177.             }
178.          });
179.    }
180. }
```

API javax.swing.JFileChooser 1.2

- `JFileChooser()`
  创建一个可用于多框架的文件选择器对话框。
- `void setCurrentDirectory(File dir)`
  设置文件对话框的初始目录。
- `void setSelectedFile(File file)`
- `void setSelectedFiles(File[] file)`
  设置文件对话框的默认文件选择。
- `void setMultiSelectionEnabled(boolean b)`
  设置或清除多选模式。
- `void setFileSelectionMode(int mode)`
  设置用户选择模式，只可以选择文件（默认），只可以选择目录，或者文件和目录均可

以选择。mode参数的取值可以是JFileChooser.FILES_ONLY、JFileChooser.DIRECTORIES_ONLY和JFileChooser.FILES_AND_DIRECTORIES 之一。

- `int showOpenDialog(Component parent)`
- `int showSaveDialog(Component parent)`
- `int showDialog(Component parent, String approveButtonText)`

显示按钮标签为Open，Save或者approveButtonText字符串的对话框，并返回APPROVE_OPTION、CANCEL_OPTION （如果用户选择取消按钮或者离开了对话框）或者ERROR_OPTION（如果发生错误）。

- `File getSelectedFile()`
- `File[] getSelectedFiles()`

获取用户选择的一个文件或多个文件（如果用户没有选择文件，返回null）。

- `void setFileFilter(FileFilter filter)`

设置文件对话框的文件过滤器。所有让filter.accept返回true 的文件都会被显示，并且将过滤器添加到可选过滤器列表中。

- `void addChoosableFileFilter(FileFilter filter)`

将文件过滤器添加到可选过滤器列表中。

- `void setAcceptAllFileFilterUsed(boolean b)`

在过滤器组合框中包括或者取消All files过滤器。

- `void resetChoosableFileFilters()`

清除可选过滤器列表。除非All files过滤器被显式地清除，否则它仍然会存在。

- `void setFileView(FileView view)`

设置一个文件视图来提供文件选择器显示信息。

- `void setAccessory(JComponent component)`

设置一个附件组件。

API `javax.swing.filechooser.FileFilter` 1.2

- `boolean accept(File f)`

如果文件选择器可以显示这个文件，返回true。

- `String getDescription()`

返回这个文件过滤器的说明信息，例如，Image files (*.gif,*.jpeg)。

API `javax.swing.filechooser.FileNameExtensionFiler` 6

- `FileNameExtensionFilter(String description, String ... extensions)`

利用给定的描述构造一个文件过滤器。这些描述限定了被接受的所有目录和文件其名称结尾的句点之后所包含的扩展字符串。

API `javax.swing.filechooser.FileView` 1.2

- `String getName(File f)`

返回文件f的文件名，或者null。通常这个方法返回f.getName()。

- `String getDescription(File f)`
  返回文件f的可读性描述，或者null。例如，如果f是HTML文档，那么这个方法有可能返回它的标题。
- `String getTypeDescription(File f)`
  返回文件f的类型的可读性描述。例如，如果f是HTML文档，那么这个方法有可能返回Hypertext document字符串。
- `Icon getIcon(File f)`
  返回文件f的图标，或者null。例如，如果f是JPEG文件，那么这个方法有可能返回简略的图标。
- `Boolean isTraversable(File f)`
  如果f是用户可以打开的目录，返回Boolean.TRUE。如果目录在概念上是复合文档，那么这个方法有可能返回false。与所有的FileView 方法一样，这个方法有可能返回null，用于表示文件选择器应该使用默认视图。

9.7.5 颜色选择器

前面曾经说过，一个高质量的文件选择器是一个很复杂的用户界面组件，人们肯定不愿意自己去编写它。许多用户界面工具包还提供了另外一些常用的对话框：选择日期/时间、货币值、字体以及颜色等。这将会带来两个方面的好处：程序员可以直接地使用这些高质量的代码而不用从头做起，并且用户通常具有使用这些选择的经验。

除了文件选择器外，Swing还提供了一种选择器——JColorChooser（如图9-42～图9-44）。可以利用这个选择器选取颜色。与JFileChooser一样，颜色选择器也是一个组件，而不是一个对话框，但是它包含了用于创建包含颜色选择器组件的对话框方法。

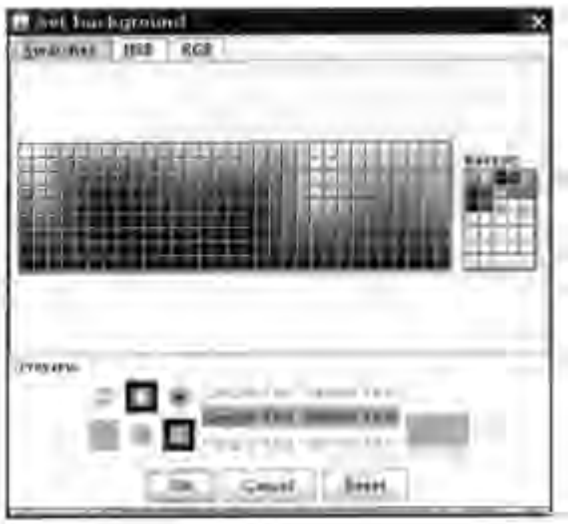


图9-42 颜色选择器的swatches窗格

图9-43 颜色选择器的HSB窗格

图9-44 颜色选择器的RGB窗格

下面这段代码说明了如何利用颜色选择器显示模式对话框：

```
Color selectedColor = JColorChooser.showDialog(parent,title, initialColor);
```

另外，也可以显示无模式颜色选择器对话框，需要提供：

- 一个父组件。
- 对话框的标题。
- 选择模式/无模式对话框的标志。
- 颜色选择器。
- OK和Cancel按钮的监听器（如果不需要监听器可以设置为null）。

下面这段代码将会创建一个无模式对话框。当用户按下OK键时，对话框的背景颜色就会被设成所选择的颜色。

```
chooser = new JColorChooser();
dialog = JColorChooser.createDialog(
   parent,
   "Background Color",
false /* not modal */,
chooser,
new ActionListener() // OK button listener
   {
      public void actionPerformed(ActionEvent event)
      {
         setBackground(chooser.getColor());
      }
   },
null /* no Cancel button listener */);
```

读者还可以做进一步的改进，将颜色选择立即反馈给用户。如果想要监视颜色的选择，那就需要获得选择器的选择模型并添加改变监听器：

```
chooser.getSelectionModel().addChangeListener(new
   ChangeListener()
```

```
{
   public void stateChanged(ChangeEvent event)
   {
      do something with chooser.getColor();
   }
});
```

在这种情况下，颜色选择器对话框提供的OK和Cancel没有什么用途。可以将颜色选择器组件直接添加到一个无模式对话框中：

```
dialog = new JDialog(parent, false /* not modal */);
dialog.add(chooser);
dialog.pack();
```

例9-18的程序展示了三种对话框类型。如果点击Model（模式）按钮，则需要选择一个颜色才能够继续后面的操作。如果点击Modaless（无模式）按钮，则会得到一个无模式对话框。这时只有点击对话框的OK按钮时，颜色才会发生改变。如果点击Immediate按钮，那将会得到一个没有按钮的无模式对话框。只要选择了对话框中的一个颜色，面板的背景就会立即更新。

例9-18 ColorChooserTest.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import javax.swing.*;
4. import javax.swing.event.*;
5.
6. /**
7.  * @version 1.03 2007-06-12
8.  * @author Cay Horstmann
9.  */
10. public class ColorChooserTest
11. {
12.    public static void main(String[] args)
13.    {
14.       EventQueue.invokeLater(new Runnable()
15.          {
16.             public void run()
17.             {
18.                ColorChooserFrame frame = new ColorChooserFrame();
19.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.                frame.setVisible(true);
21.             }
22.          });
23.    }
24. }
25.
26. /**
27.  * A frame with a color chooser panel
28.  */
29. class ColorChooserFrame extends JFrame
30. {
31.    public ColorChooserFrame()
32.    {
33.       setTitle("ColorChooserTest");
34.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
```

```
35.
36.       // add color chooser panel to frame
37.
38.       ColorChooserPanel panel = new ColorChooserPanel();
39.       add(panel);
40.    }
41.
42.    public static final int DEFAULT_WIDTH = 300;
43.    public static final int DEFAULT_HEIGHT = 200;
44. }
45.
46. /**
47.  * A panel with buttons to pop up three types of color choosers
48.  */
49. class ColorChooserPanel extends JPanel
50. {
51.    public ColorChooserPanel()
52.    {
53.       JButton modalButton = new JButton("Modal");
54.       modalButton.addActionListener(new ModalListener());
55.       add(modalButton);
56.
57.       JButton modelessButton = new JButton("Modeless");
58.       modelessButton.addActionListener(new ModelessListener());
59.       add(modelessButton);
60.
61.       JButton immediateButton = new JButton("Immediate");
62.       immediateButton.addActionListener(new ImmediateListener());
63.       add(immediateButton);
64.    }
65.
66.    /**
67.     * This listener pops up a modal color chooser
68.     */
69.    private class ModalListener implements ActionListener
70.    {
71.       public void actionPerformed(ActionEvent event)
72.       {
73.          Color defaultColor = getBackground();
74.          Color selected = JColorChooser.showDialog(ColorChooserPanel.this, "Set background",
75.                defaultColor);
76.          if (selected != null) setBackground(selected);
77.       }
78.    }
79.
80.    /**
81.     * This listener pops up a modeless color chooser. The panel color is changed when the user
82.     * clicks the Ok button.
83.     */
84.    private class ModelessListener implements ActionListener
85.    {
86.       public ModelessListener()
87.       {
88.          chooser = new JColorChooser();
89.          dialog = JColorChooser.createDialog(ColorChooserPanel.this, "Background Color",
90.                false /* not modal */, chooser, new ActionListener() // OK
91.                // button
```

```
92.                // listener
93.                   {
94.                      public void actionPerformed(ActionEvent event)
95.                      {
96.                         setBackground(chooser.getColor());
97.                      }
98.                   }, null /* no Cancel button listener */);
99.       }
100.
101.      public void actionPerformed(ActionEvent event)
102.      {
103.         chooser.setColor(getBackground());
104.         dialog.setVisible(true);
105.      }
106.
107.      private JDialog dialog;
108.      private JColorChooser chooser;
109.   }
110.
111.   /**
112.    * This listener pops up a modeless color chooser. The panel color is changed immediately
113.    * when the user picks a new color.
114.    */
115.   private class ImmediateListener implements ActionListener
116.   {
117.      public ImmediateListener()
118.      {
119.         chooser = new JColorChooser();
120.         chooser.getSelectionModel().addChangeListener(new ChangeListener()
121.            {
122.               public void stateChanged(ChangeEvent event)
123.               {
124.                  setBackground(chooser.getColor());
125.               }
126.            });
127.
128.         dialog = new JDialog((Frame) null, false /* not modal */);
129.         dialog.add(chooser);
130.         dialog.pack();
131.      }
132.
133.      public void actionPerformed(ActionEvent event)
134.      {
135.         chooser.setColor(getBackground());
136.         dialog.setVisible(true);
137.      }
138.
139.      private JDialog dialog;
140.      private JColorChooser chooser;
141.   }
142. }
```

API javax.swing.JColorChooser 1.2

- JColorChooser()

构造一个初始颜色为白色的颜色选择器。

- `Color getColor()`
- `void setColor(Color c)`

获取和设置颜色选择器的当前颜色。

- `static Color showDialog(Component parent, String title, Color initialColor)`

显示包含颜色选择器的模式对话框。

参数：	parent	对话框显示在上面的组件
	title	对话框框架的标题
	initialColor	颜色选择器的初始颜色

- `static JDialog createDialog(Component parent, String title, boolean modal, JColorChooser chooser, ActionListener okListener, ActionListener cancelListener)`

创建一个包含颜色选择器的对话框。

参数：	parent	对话框显示在上面的组件
	title	对话框框架的标题
	modal	如果直到对话框关闭，这个调用都被阻塞，则返回true
	chooser	添加到对话框中的颜色选择器
	okListener, cancelListener	OK和Cancel按钮的监听器

至此将结束用户界面组件的讨论。第7章～第9章讲述了如何用Swing实现简单的图形用户界面。卷II将讨论更加高级的Swing组件和更加复杂的图形技术。