本章内容：

- 介绍 Swing 菜单
- 菜单栏选择模型
- JMenuBar 类
- JMenuItem 类
- JPopupMenu 类
- JMenu 类
- 可选菜单项
- 工具栏

第十四章

菜单和工具栏

本章讨论 Swing 菜单和工具栏。由于菜单较之工具栏内容更丰富、功能更灵活，所以占据了本章大部分篇幅。用户在学习一个新应用程序时常常先从菜单和工具栏入手，因此，Swing 为菜单组件的设计提供大量的自由空间。

工具栏允许程序员将按钮和其他元素共同组合到一块支持重定位的面板里，其中的工具可以帮助用户完成许多常规任务。用户可以往 Swing 工具栏中添加任意组件，甚至连非 Swing 组件也可以。此外，为方便起见，Swing 还允许将工具栏从框架拖出，放到子窗口中。

介绍 Swing 菜单

Swing 菜单组件是 `JComponent` 的子类，因而拥有 Swing 组件的所有优点，程序员可以将之当做布局管理器或容器看待。

下面列出了 Swing 菜单系统的一些显著特性：

- 图标可以扩充或替换菜单项。
- 菜单项可以为单选按钮。
- 可以给菜单项指定快捷方式（accelerator），它们与菜单项文字相邻。
- 大多数标准 Swing 组件可用做菜单项。

Swing 为应用程序提供大家熟悉的菜单分隔线、复选框菜单项、弹出式菜单和子菜单。此外，Swing 菜单还支持快捷方式和下划线式快捷方式（助记符），并可以通过适当调整框架的镶边，把菜单栏置于 Swing 框架顶部。在 Macintosh 上，用户可以对应用程序进行配置，以便按自己意愿把菜单栏放到屏幕顶部。图 14-1 定义了组成 Swing 菜单系统的各种元素。

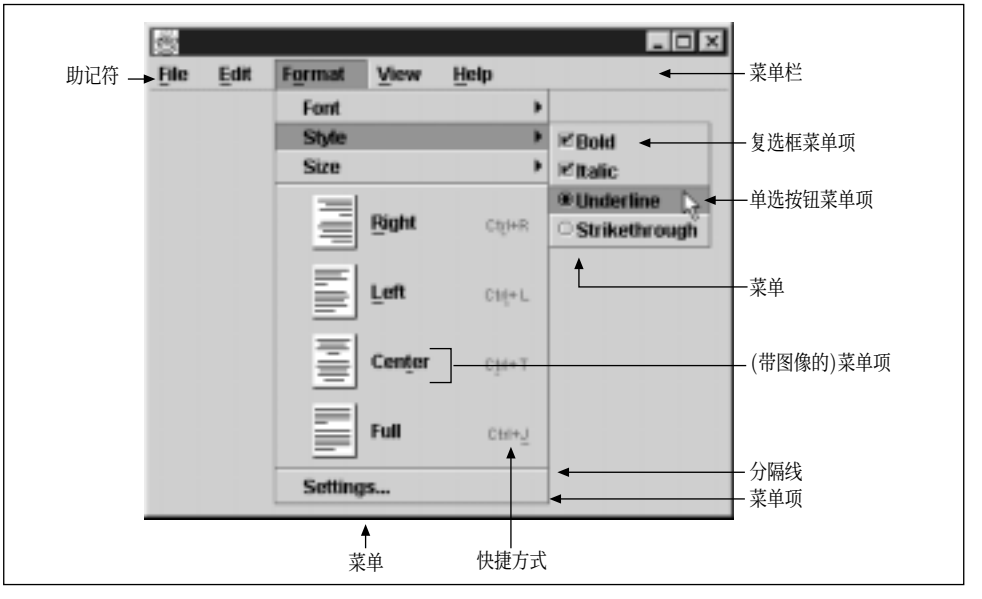


图 14-1：Swing 菜单系统的元素

请注意，下划线式的助记符并不被所有平台所支持。事实上，Macintosh 的系统菜单栏中就没有这种助记符（它从不提供这种用户界面），即使菜单中出现，该助记符也不起作用。因此，如果应用程序运用助记符，就要考虑将该部分代码组建成一个独立的方法，只有当所运行的平台支持它们时，才被调用。

所有平台都支持快捷方式（快捷键），但在快捷方式的调用键上各自的约定不尽相同。`Toolkit` 方法 `getMenuShortcutKeyMask` 可保证程序员总是使用正确的调用键。

菜单层次结构

图 14-2 是 Swing 菜单的类图解。

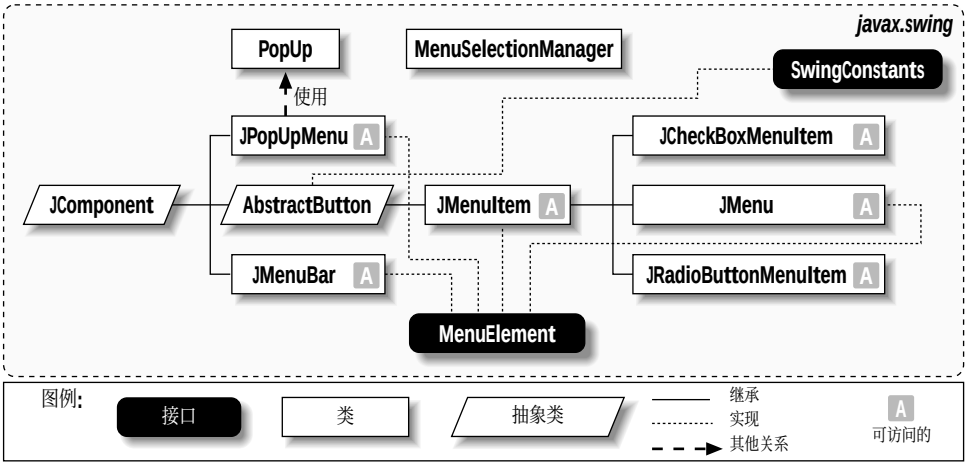


图 14-2：Swing 菜单图解

读者会惊奇地发现该层次结构中竟然也包括AbstractButton，确实，菜单和菜单项包含许多与 Swing 按钮相同的特性。举例来说，菜单项可以在鼠标指针经过时高光显示，可以被单击以表明用户做出了选择，可以像按钮一样被禁止而呈现灰色，还可以被指定动作指令（action command）以协助事件处理。其中的JCheckBoxMenuItem和JRadioButtonMenuItem对象甚至可以在两个选择状态之间切换。既然 Swing 菜单组件共享了大量 Swing 按钮的功能，那么从AbstractButton继承菜单组件就不失为一种妥当高效的做法。

还有一件出人意料的事，JMenu类继承自JMenuItem类，反之不然。这是因为每个JMenu对象都包含一个用作菜单标题的隐含菜单项，常被称做标题按钮（title button）。当用户按下鼠标或拖动鼠标光标经过标题按钮时，其对应的菜单便会显示出来。但必须注意的是，菜单不必锚定在菜单栏上，它们可以被嵌入其他菜单中充当子菜单。这就是说，标题按钮必须起到菜单项的作用，这一点在层次结构被反过来时会难以实现。本章稍后论及JMenu类时会深入讨论此特性。

几乎所有的菜单类都实现MenuElement接口。此接口概括地描述了每一个Swing菜单组件在遇到用户输入（比如键盘事件或鼠标事件）时如何应对的标准化方法。一般情况下，Swing 菜单类会处理上述鼠标和键盘事件，并通知组件代理（component delegate）完成必要的组件重画操作。上述方法与MenuSelectionManager类协同工作。虽然很少需要程序员亲自实现MenuElement接口，但知道其工作方法总是有益无弊。本章稍后将示范此接口的实现方法。

小试 Swing 菜单

现在，让我们来亲自使用 Swing 菜单的功能。以下的程序将向读者介绍许多基本的 Swing 菜单功能。

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

public class IntroExample extends JMenuBar {

    String[] fileItems = new String[] { "New", "Open", "Save", "Exit" };
    String[] editItems = new String[] { "Undo", "Cut", "Copy", "Paste" };
    char[] fileShortcuts = { 'N','O','S','X' };
    char[] editShortcuts = { 'Z','X','C','V' };

    public IntroExample() {

        JMenu fileMenu = new JMenu("File");
        JMenu editMenu = new JMenu("Edit");
        JMenu otherMenu = new JMenu("Other");
        JMenu subMenu = new JMenu("SubMenu");
        JMenu subMenu2 = new JMenu("SubMenu2");

        // Assemble the File menus with mnemonics.
        ActionListener printListener = new ActionListener() {
                public void actionPerformed(ActionEvent event) {
                    System.out.println("Menu item [" + event.getActionCommand() +
                          "] was pressed.");
                }
            };
        for (int i=0; i < fileItems.length; i++) {
            JMenuItem item = new JMenuItem(fileItems[i], fileShortcuts[i]);
            item.addActionListener(printListener);
            fileMenu.add(item);
        }

        // Assemble the File menus with keyboard accelerators.
        for (int i=0; i < editItems.length; i++) {
            JMenuItem item = new JMenuItem(editItems[i]);
            item.setAccelerator(KeyStroke.getKeyStroke(editShortcuts[i],
                Toolkit.getDefaultToolkit().getMenuShortcutKeyMask(), false));
            item.addActionListener(printListener);
            editMenu.add(item);
        }

        // Insert a separator in the Edit menu in Position 1 after "Undo".
        editMenu.insertSeparator(1);

        // Assemble the submenus of the Other menu.
```

```
            JMenuItem item;
            subMenu2.add(item = new JMenuItem("Extra 2"));
            item.addActionListener(printListener);
            subMenu.add(item = new JMenuItem("Extra 1"));
            item.addActionListener(printListener);
            subMenu.add(subMenu2);

            // Assemble the Other menu itself.
            otherMenu.add(subMenu);
            otherMenu.add(item = new JCheckBoxMenuItem("Check Me"));
            item.addActionListener(printListener);
            otherMenu.addSeparator();
            ButtonGroup buttonGroup = new ButtonGroup();
            otherMenu.add(item = new JRadioButtonMenuItem("Radio 1"));
            item.addActionListener(printListener);
            buttonGroup.add(item);
            otherMenu.add(item = new JRadioButtonMenuItem("Radio 2"));
            item.addActionListener(printListener);
            buttonGroup.add(item);
            otherMenu.addSeparator();
            otherMenu.add(item = new JMenuItem("Potted Plant",
                                  new ImageIcon("image.gif")));
            item.addActionListener(printListener);

            // Finally, add all the menus to the menu bar.
            add(fileMenu);
            add(editMenu);
            add(otherMenu);
        }

        public static void main(String s[]) {
            JFrame frame = new JFrame("Simple Menu Example");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setJMenuBar(new IntroExample());
            frame.pack();
            frame.setVisible(true);
        }
    }
```

上例创建了一个包含三个菜单的菜单栏，File菜单的菜单项带有助记符，Edit菜单的菜单项带有快捷方式。图14-3为程序生成的三个不同菜单以及Edit菜单在两种不同平台上的显示效果，这两种平台的快捷键（accelerator key）分别为Control和Command。

第三个菜单的末项旁添加了一个GIF格式的盆景图像。此外，Other菜单的第一个菜单项实际上是一个子菜单，它会弹出另一个子菜单，说明了菜单的递归特性。选择其中任何一个菜单，将出现一段文字说明当前单击的内容：

```
Menu item [New] was pressed.
Menu item [Radio 1] was pressed.
```

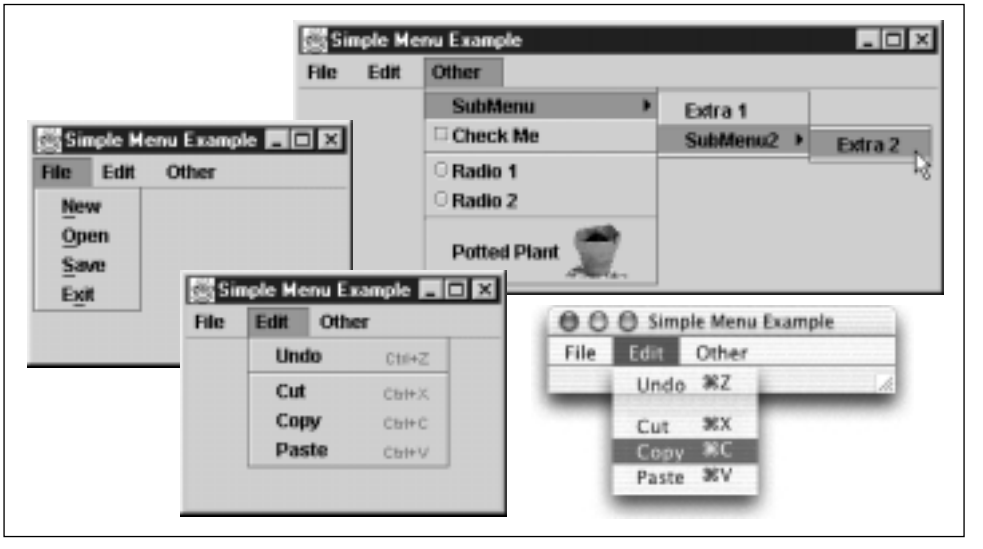


图 14-3：Swing 菜单效果示例

如果读者现在对上面提及的类和方法不甚明了，也不必担心。本书很快就会对每一个菜单组件做详细的讲解。

菜单栏选择模型

所有的 GUI 环境都只允许菜单组件每次做一个选择，Swing 组件也不例外。它提供了数据模型 `SingleSelectionModel` 供菜单栏和菜单模拟上述特性。

SingleSelectionModel 接口

`SingleSelectionModel`接口的实现对象完成的功能正如其名：它们保持一组可供选择的元素，但每次只允许用户选择其中的一个。所有被选元素的索引都存储在该模型中，一旦出现新的被选元素，模型就重置代表被选元素的索引，并触发每个注册监听器的 `ChangeEvent` 事件。

属性

`SingleSelectionModel`接口的实现对象包含的属性如表14-1所示。其中，`selected`属性是`boolean`型，判断选择是否存在。`selectedIndex`属性是一个整型索引，代表当前的被选项。

表 14-1：SingleSelectionModel 的属性

属性	数据类型	get	is	set	默认值
selected	boolean		•		
selectedIndex	int	•		•	

事件

SingleSelectionModel接口的实现对象在修改selectedIndex属性时，例如改变选择时，必须触发一个 ChangeEvent 事件（而非 PropertyChangeEvent 事件）。为了维护 ChangeEvent 监听器列表，接口包含了标准 addChangeListener()方法和 removeChangeListener()方法。

void addChangeListener(ChangeListener listener)

void removeChangeListener(ChangeListener listener)

从接收模型变化事件的监听器列表中添加或删除指定的 ChangeListener。

方法

SingleSelectionModel 接口也包含一个方法：

public void clearSelection()

清除选定值，强制 selected 属性返回 false。

DefaultSingleSelectionModel 类

DefaultSingleSelectionModel类是Swing对SingleSelectionModel接口的默认实现。

属性

DefaultSingleSelectionModel仅仅包含了SingleSelectionModel接口的必需属性，如表 14-2 所示。其中，selectedIndex 属性是一个整型索引，代表当前的被选项。默认值 -1 显示当前没有项目被选中。selected 属性是 boolean 型，当 selectedIndex 为 -1 时它返回 false，反之返回 true。

表 14-2：DefaultSingleSelectionModel 的属性

属性	数据类型	get	is	set	默认值
selected	boolean		•		false
selectedIndex	int	•		•	-1

事件和方法

之前讨论的SingleSelectionModel接口指定的所有事件和方法都由DefaultSingle-SelectionModel对象提供。

JMenuBar 类

Swing的JMenuBar类是AWT的MenuBar类的替代产品，它创建一个水平菜单栏组件，其中可连接零个以上菜单。JMenuBar 类将DefaultSingleSelectionModel用做自身的数据模型，因为用户在某一时刻只能唤起（raise）或激活（activate）一个菜单。而且，只要鼠标指针离开菜单，该类就把菜单从屏幕上去除（或按照Swing说法，将其“ 取消 ”),所有菜单又恢复为可以被唤起的状态。图14-4显示了JMenuBar组件的类层次结构（class hierarchy)。

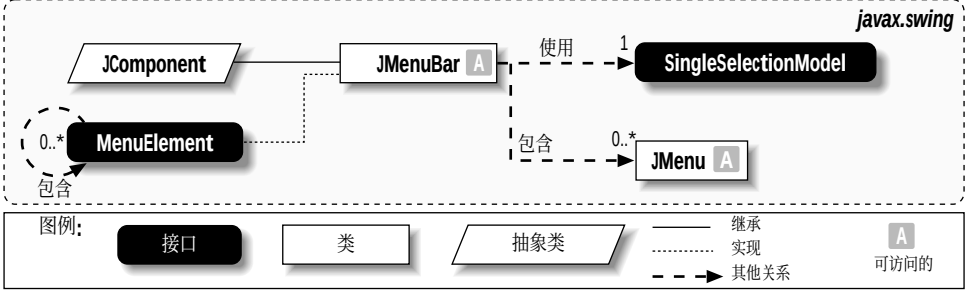


图 14-4：JMenuBar 类图

程序员可以用 JMenuBar 类的 add() 方法添加 JMenu 对象。JMenuBar 便随之分配一个整数索引，里面的菜单按照其加入索引的顺序排列，而菜单栏则依照此索引内容的顺序从左至右显示菜单。从理论上讲，帮助菜单是惟一的例外。程序员应该让某一个菜单被标记为帮助菜单，但帮助菜单的位置由外观风格决定。在实践中，如果程序员越俎代庖， JMenuBar 就会抛出一个 Error 对象。

菜单栏布局

有两种方法可以将菜单栏连接到 Swing 框架或 applet 上。第一，使用 JFrame、JDialog、JApplet 或 JInternalFrame 的 setJMenuBar()方法：

```
JFrame frame = new JFrame("Menu");
JMenuBar menuBar = new JMenuBar();

//  the menu bar to the frame.
frame.setJMenuBar(menuBar);
```

这段程序中的setJMenuBar()方法类似于java.awt.Frame的setMenuBar()方法。与老版本一样，setJMenuBar()也让外观风格来确定菜单的位置（一般情况下，它会将菜单栏锚定在框架顶部，并对框架的固有 Insets 做适当调整）。JApplet 和 JDialog 都包含 setJMenuBar()方法，这也就是说，applet 和对话框中都可以添加菜单栏。不管怎样，程序员在使用 Swing 菜单时千万不要混淆 setJMenuBar()方法和老版的 AWT setMenuBar()方法，否则编译器一定吃不消。

如果应用程序在 Macintosh 上运行，用户可以设置其外观风格，按自己的意愿把菜单栏放到屏幕顶部。系统属性 com.apple.macos.useScreenMenuBar 为 true 时，该状态即被激活。但是，在默认情况下此状态是被禁止的，因为大多数 Java 程序并不期待出现这种情况，而且必须对程序适当编码才可处理该状态。尤其要注意的是，Aqua 人机界面规范（Aqua Human Interface Guidelines）要求菜单栏始终可见。如果应用程序的某个框架没有菜单栏，那么，当该窗口获得输入焦点时，菜单栏会立刻消失，让用户不知所措。解决这一问题最普遍的方法是编写一个菜单工厂（menu factory），为应用程序用到的每个框架生成一个同样的菜单栏。尽管这样做增大了程序员的工作量，但为了给 Mac 用户呈现一个亲切、舒适的界面，多做点工作还是值得的。

连接菜单栏的第二种方法是调用 JComponent 的子类 JMenuBar 类，这种方法相比之下用得不多。这意味着可以和其他 Swing 组件一样，通过 Swing 的布局管理器实现菜单栏定位。举例来说，我们可以用下列代码替换对 setJMenuBar()的调用：

```
menuBar.setBorder(new BevelBorder(BevelBorder.RAISED));
frame.getContentPane().add(menuBar, BorderLayout.SOUTH);
```

这样一来，菜单栏被置于框架底部，如图 14-5 所示（请注意，设置环绕菜单栏的斜面边框是为了突出菜单栏的位置）。甚至还可以在不同的位置添加多个菜单栏。Swing 不要求将单个菜单栏锚定在框架顶部。由于都是 JComponent 的子类，所以多重（multiple）菜单栏可以位于容器内的任何位置。

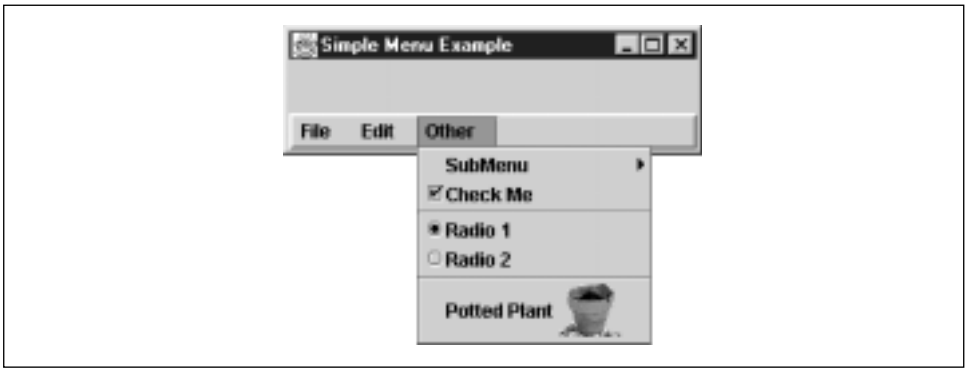


图 14-5：按 Swing 组件的方式定位 JMenuBar

注意：程序员必须至少给菜单栏添加一个有名称的菜单，这样菜单栏才具备一定的宽度。否则，它将显示为一条与分隔线相似的细线。

当然，如果没有强制性的理由，程序员是不会照此办理的。但这样就影响了外观风格在适当的时机给菜单栏妥善定位。移动菜单栏这类基础组件，几乎总是会让用户手忙脚乱，不知该如何操作。如果程序中还包含多重菜单栏，大概就更热闹了。

属性

`JMenuBar` 类的属性如表 14-3 所示。其中，`menu` 属性是一个索引属性（indexed property），引用了每一个连接到菜单栏的 `JMenu` 对象。只读属性 `menuCount` 则存放被连接菜单的数目。请记住，单个选择模型只允许每次激活一个菜单。如果当前有菜单被激活，`selected` 属性返回 `true`；否则返回 `false`。`componentAtIndex` 属性可访问与给定索引有关的菜单，它与被索引的 `menu` 属性相似，只是它的内容被抛给一个 `Component` 对象。如果没有组件与索引相关，`getComponentAtIndex()` 存取方法将返回 `null` 值。`component` 属性返回一个对 `this` 的引用（即菜单栏本身），`subElements` 属性返回由菜单栏中各个菜单组成的数组。

表 14-3：JMenuBar 的属性

属性	数据类型	get	is	set	默认值
`accessibleContext`[o]	`AccessibleContext`	•			`JMenuBar.AccessibleJMenuBar()`

表 14-3：JMenuBar的属性（续）

属性	数据类型	get	is	set	默认值
borderPainted[b]	boolean		•	•	true
component	Component	•			this
componentAtIndex[i]	Component	•			true
helpMenu[u]	JMenu	•		•	抛出Error
layout[o]	LayoutManager	•		•	BoxLayout(X_AXIS)
margin[b]	Insets	•		•	null
menuCount	int	•			0
menu[i]	JMenu	•			null
selected	boolean		•		false
selectionModel[b]	SingleSelection-Model	•		•	DefaultSingle-SelectionModel()
subElements	MenuElement[]	•			
UI[b]	MenuBarUI	•		•	由外观风格决定
UIClassID[o]	String	•			"MenuBarUI"

[b] 绑定，[i] 索引，[o] 覆盖，[u] 未实现

请参见表 3-6 的 JComponent 类属性

上表中的margin属性控制菜单栏边框和菜单之间的间距量。borderPainted属性用于禁止菜单栏边框的绘制，即使border属性为非null。borderPainted属性为false时，将不能实现正常的边框绘制。有关 Swing 边框的更多介绍，请参见第十三章。

警告：helpMenu属性应该允许程序员指定一个JMenu对象作为帮助菜单（它在某些操作系统中有独特的定位方式），不过，该属性从未被实现过，在 SDK 1.4 里使用该属性甚至会使系统抛出Error 对象。此外，程序员应该知道，菜单栏可以调用 BoxLayout 对象在适当时候插入“ 胶水 ”，以便把（常规的）帮助菜单置于菜单栏右端。这一方式虽然对程序员有帮助，但程序员必须编写代码以明确应该在什么时候插入“ 胶水 ”，这从而增加了编程负担。

构造函数

public JMenuBar()

创建和初始化一个空 JMenuBar 对象。

菜单

public JMenu add(JMenu menu)

该方法用于把一个JMenu对象连接到菜单栏集合中。在JMenuBar的BoxLayout对象的作用下，菜单将按程序员add()的先后顺序从左到右排列在菜单栏里。该方法返回一个对所传参数JMenu的引用，还允许程序员将调用串连起来，例如menubar.add(menu).add(menuitem)。

其他方法

public int getComponentIndex(Component c)

返回与参数所传组件引用有关的索引。如果没有与组件匹配的索引，该方法返回 -1。JMenu组件是惟一有意义的参数。

public void setSelected(Component c)

强迫菜单栏（及其相关模型）选择一个具体菜单，它将在菜单栏的单个选择模型中触发一个ChangeEvent事件。比如，当用户按下某个特定菜单的助记符键时，此方法即被调用。请注意，这与表14-3所列的boolean型selected属性并不一样。

public void updateUI()

强迫 UIManager 根据当前的UI 代理刷新组件的外观风格。

JMenuBar 还可实现 MenuElement 接口指定的方法，本章稍后将介绍。

JMenuItem 类

在讨论菜单之前,我们先介绍JMenuItem类。图14-6显示了JMenuItem组件的类图。

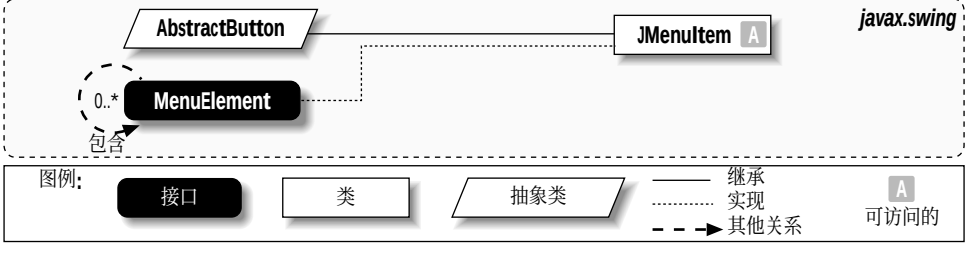


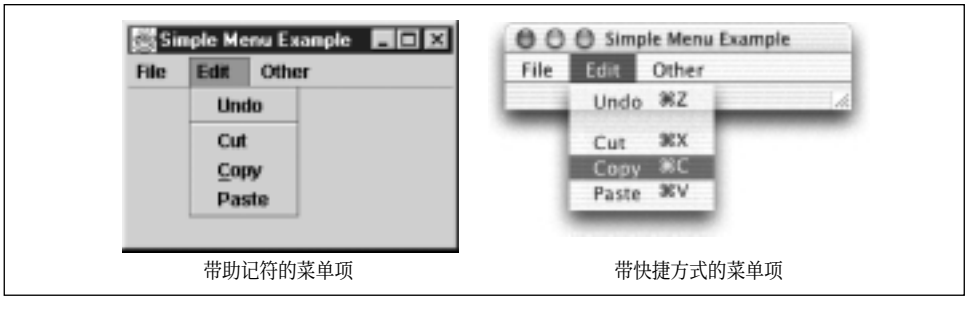
图 14-6：JMenuItem 类图

JMenuItem 是被用做菜单元素的字符串和图像的包装。它本质上是一个继承自AbstractButton类的特定按钮。不过，它的特性又不完全等同于独立按钮。当鼠标指针经过某个菜单项时，Swing就认为该菜单项被选中（selected）。当用户在该菜单项上释放鼠标键，此菜单项即被认为选中并应完成相应操作。

麻烦的是，此处存在一个术语冲突。当鼠标移到某个菜单项上时，Swing认为它被选中，如同被MenuSelectionManager类和实现MenuElement接口的类更新一样。但是，当某个按钮处于两种持续状态中的任意一种时，Swing也认为它被选中，比如，复选框按钮保持选中状态直到再次被单击。所以，当菜单项被选中时，它的按钮模型实际上也被选中（armed）。反之，当菜单项被取消选定时，它的按钮模型实际上也被取消选定（disarmed）。最后，当用户在菜单项上释放鼠标时，Swing认为按钮被单击，于是调用 AbstractButton 的 doClick()方法。

菜单项快捷键

菜单项可以兼有键盘快捷方式和（在某些平台上的）助记符。助记符是人造按钮，在代表快捷键的字符下有一条下划线。另一方面，键盘快捷方式则属于JComponent的子类。菜单项里的键盘快捷方式具有独特的作用（其确切的外观和位置由外观风格决定）。图 14-7 是助记符和键盘快捷方式的图例。



带助记符的菜单项　　带快捷方式的菜单项

图 14-7：助记符和键盘快捷方式

键盘快捷方式和助记符其实是异曲同工：都是为了让用户通过按键简化普通的GUI操作。但是，助记符只有在它所代表的按钮（或菜单项）出现在屏幕上时才能被激活，而菜单项的键盘快捷方式可以在应用程序具有输入焦点的任何时候被启用 ——不管当前菜单项是否可见。此外，快捷方式可以在所有平台和外观风格中运行，而助记符没有这么好的通用性。不管怎样，两者可同时应用于菜单。

下面来看两种方式的编程方法。键盘快捷方式一般使用多种按键：功能键、命令键或字母数字键与一到多个修饰键配合使用，比如Shift，Ctrl或Alt。所有组合键可以用javax.swing.KeyStroke类表示，但并非所有的修饰键都适合于程序运行的平台和外观风格。因此，程序员可以先用默认工具箱的menuShortcutKeyMask属性配置KeyStroke对象，再用KeyStroke对象配置键盘快捷方式的accelerator属性，并把此属性赋予某个菜单项。

```
JMenuItem m = new JMenuItem("Copy");
m.setAccelerator(KeyStroke.getKeyStroke('C',
    Toolkit.getDefaultToolkit().getMenuShortcutKeyMask(), false));
```

在Metal外观风格下，这段代码将Ctrl-C设定为快捷方式，即同时按下字母键C和Ctrl键。快捷方式位于菜单项右边（此位置也是由外观风格决定的）。关于KeyStroke类的内容将在本书第二十七章详细讲述。

另一种通用性稍弱的设置快捷键的方法是通过AbstractButton超类的mnemonic属性完成的：

```
JMenuItem mi = new JMenuItem("Copy");
mi.setMnemonic('C');
```

mnemonic属性在传递给setMnemonic()方法的参数字符下加划线。请注意，助记符字符没有修饰键，它们都是单一字母。而且，助记符一定要使用菜单项标签中的字母，否则，菜单项中没有带下划线的字母，用户也就不知道怎样激活快捷键。最后，请确保程序的运行平台和外观风格支持助记符功能。

在SDK 1.4中，程序员可以使用displayedMnemonicIndex属性处理菜单项中的多个助记符字母。例如，Save As菜单项中应该在大写的“A”下面加下划线，为了做到这一点，必须在确立助记符后立刻将setDisplayedMnemonicIndex赋值为5。

图像

Swing的菜单项可以包含图标或完全由图标构成。在图标能够清楚传达含义的情况下，这样做无疑是非常直观的。程序员可以给JMenuItem类的构造函数传递一个Icon对象，如下所示：

```
JMenu menu = new JMenu("Justify");

// The first two menu items contain text and an image. The third
// uses only the image.
menu.add(new JMenuItem("Center", new ImageIcon("center.gif")));
```

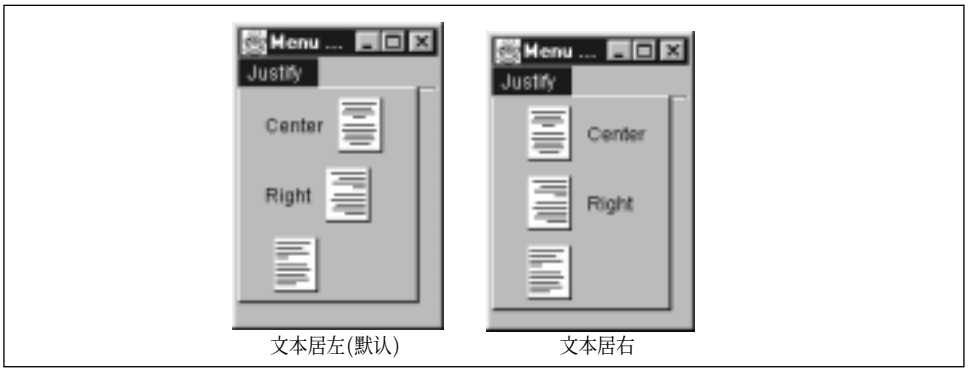
```
menu.add(new JMenuItem("Right", new ImageIcon("right.gif")));
menu.add(new JMenuItem(new ImageIcon("left.gif")));
```

在默认情况下，文字位于图像左侧，如图14-8左图所示。从图中可见，这种显示方式常常使文字右侧的图像产生位置偏移，当菜单项只由一个图像组成时尤为如此。当菜单项中的图像宽度相同时，程序员可以用setHorizontalTextAlignment()方法改变文字的位置，改善菜单的外观。

```
JMenu menu = new JMenu("Justify");

// The first two menu items contain text and an image. The third
// uses only the image. The text is now set to the right.
JMenuItem item1= new JMenuItem("Center", new ImageIcon("center.gif")));
item1.setHorizontalTextAlignment(SwingConstants.RIGHT);
JMenuItem item2= new JMenuItem("Right", new ImageIcon("right.gif")));
item2.setHorizontalTextAlignment(SwingConstants.RIGHT);

// Now add the menu items to the menu.
menu.add(item1);
menu.add(item2);
menu.add(new JMenuItem(new ImageIcon("left.gif")));
```



文本居左(默认)　　文本居右

图 14-8：菜单项的图像和文字布局

这段代码将文字置于图像右侧，如图14-8右图所示。setHorizontalTextAlignment()方法可以一直回溯到类层次结构的AbstractButton类。正如前文所述，JMenuItem类是一个关于文字和图像的按钮对象。由于AbstractButton中也包含setVerticalTextAlignment()方法，因此，如果菜单项中的图像比文字高，程序员就可以用本方法设置文字的垂直位置（关于菜单项和按钮的对齐方式，请参见第五章的AbstractButton类和第十一章的OverlayLayout类）。如果菜单项是用Action对象构造的，图像就会移到文字左侧（本章稍后将介绍更多相关内容）。

Java 支持图像透明，因此，如果程序员需要图像的某些部分透明，可以在 GIF 文件中指定“透明”色（许多绘图程序都支持），或者自行创建专用滤色器，先将特定的像素色挑选出来，改变其不透明性，再将处理后的图像传递给菜单。相比之下，前者做起来更容易些。

事件处理

处理菜单项事件的方法有很多种。鉴于菜单项从 `AbstractButton` 处继承了 `ActionEvent` 功能，因而一个可行的方法便是为每个菜单项赋予一个操作指令（这通常由指定组件自动完成），并将所有菜单项连接到同一个`ActionListener`方法上。接下来，在监听器的`actionPerformed()`方法中调用事件的`getActionCommand()`方法获取产生事件的菜单项的操作指令。这一操作将告知监听器哪个菜单项已经被单击，并允许其进行适当回应。本章之前介绍的 *IntroExample.java* 和稍后将介绍的 *PopupMenuExample.java* 都使用上述方法。

另一种方法是，程序员给每个菜单项注册一个独立的 `ActionListener` 类，来判断哪个菜单项被选中。但是，Swing能做的还不止这些。最面向对象的方法是创建一个专门的 `Action` 类，让它应对用户可能向应用程序提交的所有任务，从而让程序员把实现每个动作的程序代码与动作的名称、图标、按键及其他属性捆绑为一体。程序员随后便能用此`Action`类创建菜单项，自动地设置菜单项的文字、图像、快捷方式，等等。

上述技巧在程序员希望用多种方法（如分别用工具栏和菜单入手）调用同一个操作时尤其有效。程序员可以用同样的 `Action` 实例创建菜单项和工具栏按扭，两者的标签和外观都会很合适。当应用程序需要中止某个当前不适宜的操作时，调用`Action`实例的`setEnabled`就会自动将与该操作有关的用户接口元素进行全部更新（从而对菜单项和工具栏按扭不加区分）。同样地，对操作的名称或图标等其他属性的改变也会自动地更新所有相关的用户接口组件。

尽管 SDK 1.3 之前的版本无法直接从 `Action` 构造 `JMenuItem`，但是使用添加 `Action` 到 `JMenu` 或 `JPopupMenu` 的方法可以达到同样的效果 —— 菜单自会妥善创建并配置 `JMenuItem`。

属性

`JMenuItem`类的属性如表14-4所示。其中大部分属性都是超类属性，它们被重新配

置以保证菜单项的“按钮”如菜单项一样操作。borderPainted属性永远为false，因此菜单项没有边框。focusPainted属性也为false，这是为确保焦点矩形不会出现在菜单项四周。horizontalTextPosition属性和horizontalAlignment属性都初始化为JButton.LEFT，从而使文字位于图像图标左侧，文字和图像图标位于菜单项的左侧（重新配置上述属性的内容请参见前面的例子）。

表 14-4：JMenuItem 的属性

属性	数据类型	get	is	set	默认值
accelerator[b]	KeyStroke	•		•	null
accessibleContext[o]	AccessibleContext	•			JMenuItem.AccessibleJMenuItem()
armed[b, o]	boolean		•	•	false
borderPainted[o]	boolean		•	•	false
component[o]	Component	•			
enabled[o]	boolean		•	•	true
focusPainted[o]	boolean		•	•	false
horizontalAlignment[o]	int	•		•	JButton.LEFT
horizontalTextPosition[o]	int	•		•	JButton.LEFT
menuDragMouseListeners[1.4]	MenuDragMouseListener[]	•			
menuKeyListeners[1.4]	MenuKeyListener[]	•			
model[o]	ButtonModel	•		•	DefaultButtonModel()
subElements[o]	MenuElement[]	•			
UI[b]	MenuItemUI			•	由外观风格决定
UIClassID[o]	String	•			"MenuItemUI"

[1.4] 1.4 版以后，[b] 绑定，[o] 覆盖

请参见表 5-4 的 AbstractButton 类属性

在表 14-4 所列的属性中，accelerator 属性设置菜单项的键盘快捷方式，快捷方式一般位于菜单项字符串的右侧。armed 属性以一个 boolean 值对应 ButtonModel 组件模型的选中状态（armed state），必要时，程序员可以用此布尔值编程选择菜单项。boolean 型的 enabled 属性指示用户是否可选择菜单项，当菜单项禁用时，

JMenuItem会自动地将文字及其图像置为灰色。如前所述，对菜单项允许状态进行操控的最有效方法是将菜单项关联到一个Action对象上，这样一来，它便会自动跟踪操作的允许状态。subElements属性用于提供菜单项所包含的一组子菜单。

构造函数

JMenuItem()

JMenuItem(Action action)

JMenuItem(Icon icon)

JMenuItem(String string)

JMenuItem(String string, Icon icon)

JMenuItem(String string, int mnemonic)

使用适当的图标或字符串创建一个菜单项。如果用字符串初始化，还可以指定一个助记符。从1.3版以后，程序员可以用Action对象的属性直接配置JMenuItem的属性。

事件

JMenuItems发送的事件有很多种。其中最重要的是ActionEvents，它在菜单项被选中时被触发。ChangeEvents在按钮属性变化时被触发。为上述事件添加或删除监听器的方法都继承自AbstractButton。

JMenuItem还运用特别事件MenuDragMouseEvent和MenuKeyEvent报告菜单项顶部的鼠标移动和按键操作。下面给出为上述事件注册监听器的方法：

addMenuDragMouseListener (MenuDragMouseListener 1)

removeMenuDragMouseListener (MenuDragMouseListener 1)

上述方法添加或删除一个指定的MenuDragMouseListener，如果存在MenuDragMouseEvent，MenuDragMouseListener将被通知。

addMenuKeyListener (MenuKeyListener 1)

removeMenuKeyListener (MenuKeyListener 1)

上述方法添加或删除一个指定的MenuKeyListener，如果存在MenuKeyEvent，MenuDragMouseListener将被通知。

下面的方法用于触发上述事件，但程序员或许永远不需要调用它们：

public void processMenuDragMouseEvent (MenuDragMouseEvent e)

根据观察到的MouseEvent类型触发一个MenuDragMouseEvent通知。例如，当MouseEvent是MOUSE_ENTERED时，菜单将调用fireMenuDragMouseEntered()方法。

public void processMenuKeyEvent (MenuKeyEvent e)

根据观察到的MenuKeyEvent类型触发一个MenuKeyEvent通知。例如，当MenuKeyEvent是KEY_RELEASED时，菜单将调用fireMenuKeyReleased()方法。

方法

public void updateUI()

强迫当前的UI管理器重置当前的组件代理，从而更新组件的外观风格。

菜单元素接口

public void menuSelectionChanged(boolean isIncluded)

public MenuElement[] getSubElements()

public Component getComponent()

public void processMouseEvent(MouseEvent event, MenuElement path[], MenuSelectionManager manager)

public void processKeyEvent(KeyEvent event, MenuElement path[], MenuSelectionManager manager)

实现MenuElement接口，本章稍后将讨论。

MenuDragMouseEvent类

鼠标经过展开的菜单时，Swing会产生一系列事件。其中的MenuDragMouseEvent事件用于描述对具体菜单项的拖动。上述鼠标事件都可以被监听，方法是添加一个对象为JMenuItem的addMenuDragMouseListener()方法实现MenuDragMouseListener的监听器对象。当鼠标在菜单内拖动时，上述实现MenuDragMouseListener的监听器对象有4种不同的响应方法，每一种都会精确地表明鼠标拖动引发的后果。表14-5列出了MenuDragMouseEvent的各个属性。

属性

表 14-5：MenuDragMouseEvent 的属性

属性	数据类型	get	is	set	默认值
clickCount[o]	int	•			
id[o]	int	•		•	
manager	MenuSelectionManager	•			
modifiers[o]	Object	•		•	
path	MenuElement[]	•			
popupTrigger[o]	boolean		•		
source	Object	•			
when[o]	long	•			
x[o]	int	•			
y[o]	int	•		•	

[o] 覆盖

请参见 java.awt.event.MouseEvent

事件没有默认值，所有属性都在构造函数中设定。其中，source 属性指出发送事件的对象，id 属性描述被触发事件的类型，when 属性赋予事件时间戳（timestamp），modifiers 属性允许程序员测试各种掩码以确定按下的鼠标键是哪一个，以及 Alt、Ctrl、Shift 和 Meta 键是否被按下。x 属性和 y 属性给出鼠标指针在组件中的当前相对位置，clickCount 属性描述在鼠标拖动前鼠标键的单击次数。popupTrigger 属性指出本次鼠标事件是否会让屏幕上出现一个弹出式菜单，path 属性给出一个 MenuElement 对象的有序数组，描述通向指定菜单的路径。最后，manager 属性包含一个对菜单系统当前 MenuSelectionManager 的引用。

构造函数

public MenuDragMouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger, MenuElement[] path, MenuSelectionManager manager)

用给定值初始化表 14-5 中的所有属性。

MenuDragMouseListener 接口

作为MenuDragMouseEvent对象接收管道的MenuDragMouseListener接口共包含4种方法。第一种方法当鼠标在菜单项内部拖动时被调用，第二种方法当鼠标在菜单项内部被释放时调用。另外两种方法则分别在鼠标被拖入或拖出菜单项时调用。

方法

public abstract void menuDragMouseDragged(PopupMenuEvent e)

在菜单项内拖动鼠标时被调用。

public abstract void menuDragMouseReleased(PopupMenuEvent e)

在菜单项内释放鼠标时被调用。

public abstract void menuDragMouseEntered(PopupMenuEvent e)

当鼠标被拖动，且已进入某个菜单项时被调用。

public abstract void menuDragMouseExited(PopupMenuEvent e)

当鼠标被拖动，且已离开某个菜单项时被调用。

MenuKeyEvent 类

当特定菜单项接收一个按键事件（key event）时，Swing 也会产生一个事件。请注意，按键事件不必针对某个特定的菜单（即快捷方式或助记符），相反，菜单项会在其菜单在屏幕上弹出时响应任何一个按键事件。上述按键事件都可以被监听，方法是添加一个对象为JMenuItem的addMenuKeyListener()方法实现MenuKeyListener。实现 MenuKeyListener 的监听对象也有三种响应菜单按键事件的不同方法。

表 14-6 列出了MenuKeyEvent 的各个属性。事件没有默认值，所有属性都在构造函数中设定。其中，source 属性指出发送事件的对象，id 属性描述被触发事件的类型，when属性赋予事件一个时间戳（timestamp），modifiers属性允许程序员测试各种掩码以确定按下的是哪一个鼠标键，以及 Alt、Ctrl、Shift 和 Meta 键是否被按下。keyCode 属性和 keyChar 属性描述实际被按下的键，path 属性给出一个MenuElement 对象的有序数组，描述通向指定菜单的路径。最后，manager属性包含一个对菜单系统当前 MenuSelectionManager 的引用。

表 14-6：MenuKeyEvent 的属性

属性	数据类型	get	is	set	默认值
id[o]	int	•		•	
keyChar[o]	char	•		•	
keyCode[o]	int	•			
manager	MenuSelectionManager	•			
modifiers[o]	Object	•		•	
path	MenuElement[]	•			
source	Object	•			
when[o]	long	•			

[o] 覆盖

请参见 java.awt.event.keyEvent

构造函数

public MenuDragMouseEvent(Component source, int id, long when, int keyCode, char keyChar, MenuElement[] path, MenuSelectionManager manager)

此构造函数设置表 14-6 中的每一个属性。

MenuKeyListener 接口

作为 MenuKeyEvent 对象接收管道的 MenuKeyListener 接口共包含 3 种方法。第一种方法在某个键被敲打（typed）时（即按下并释放）调用，第二种方法在键被按下后调用，第三种在键被释放时调用。请注意，当某个键的按下状态维持了数秒，Swing 就认为该键被再次“敲打”和“按下”。

方法

public abstract void menuKeyTyped(MenuKeyEvent e)

当为菜单元素设计的键被按下和释放时调用此方法。

public abstract void menuKeyPressed(MenuKeyEvent e)

当为菜单元素设计的键被按下时调用此方法。

public abstract void menuKeyReleased(MenuKeyEvent e)

当为菜单元素设计的键被释放时调用此方法。

菜单项无法独立存在，它们必须嵌入菜单中。Swing 实现了两种风格紧密相关的菜单：锚定式菜单（anchored menu）和弹出式菜单（pop-up menu），它们分别由 JMenu 类和 JPopupMenu 类实现。

JPopupMenu 类

弹出式菜单是一种越来越常见的用户接口特色。它不连接到菜单栏，而与底层组件相关联，可自由浮动。该组件被称做调用者（invoker）。弹出式菜单与特定接口元素相连，具有很好的环境相关（context-sensitive）特性。当鼠标经过调用组件时，平台相关的（platform-dependent）弹出触发器事件会产生弹出式菜单。在 AWT 和 Swing 中，一般由鼠标事件充当触发器。菜单被唤起后，用户即可对之进行正常的交互操作。图 14-9 显示了一个 Swing 弹出式菜单。

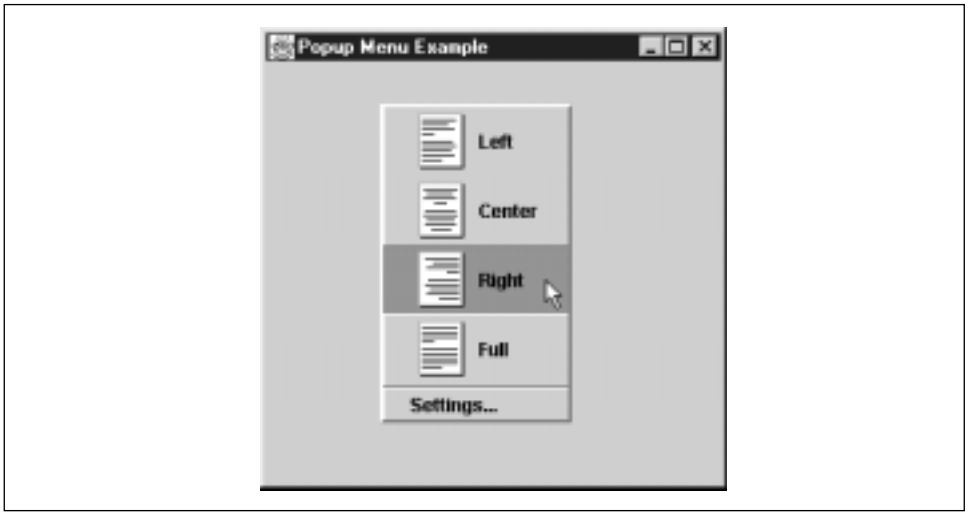


图 14-9：Swing 弹出式菜单

程序员可以使用 add()和 insert()方法添加或插入 JMenuItem、Component 或 Action 对象。其中，JPopupMenu 类给每个菜单项赋予一个整数索引，并依据弹出式菜单的布局管理器调整菜单项的顺序。此外，还可以使用 addSeparator()方法添加菜单分隔线，它们也算一种索引。图 14-10 显示了 JPopupMenu 组件的类图解。从 SDK 1.4 开始，弹出式菜单调用 Popup 类绘制自身。这个类还用于工具提示（tooltip）等简单的显示接口元素中。

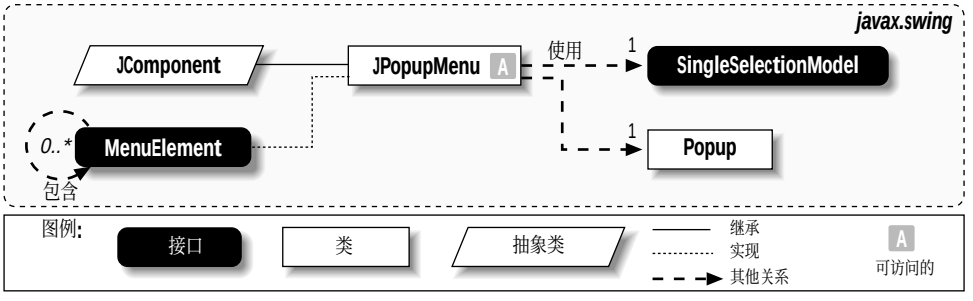


图 14-10：JPopupMenu 类图

弹出式菜单的显示

程序员常常通过调用与特定平台的弹出触发器对应的 `show()`方法来唤起弹出式菜单。`show()`方法会在菜单显示出来之前对它的 `location` 和 `invoker` 属性加以设置。弹出事件可以被各种事件自动取消，包括单击菜单项，调整调用组件的大小，或者移动、最小化、最大化或关闭父窗口（取消弹出式菜单没什么可担心的）。程序员首先检查所有的 `MouseEvents`，看它们是否是弹出触发器，然后利用触发器在合适的时候唤起弹出式菜单。请记住：不要将作为弹出触发器的 `MouseEvent` 上传给超类，否则，Swing可能会在弹出式菜单被唤起后立刻取消它。此外，请务必对按下事件和释放事件都进行检查，因为某些平台会用到它们。而解决它最简单的办法是检查所有的鼠标事件。下面的`processMouseEvent()`方法在收到适当的触发器后就唤起一个弹出式菜单：

```
public void processMouseEvent(MouseEvent e) {
    if (e.isPopupTrigger()) {
        popup.show(this, e.getX(), e.getY());
    }
    else {
        super.processMouseEvent(e);
    }
}
```

请注意，使用 `java.awt.event.MouseEvent` 中的 `isPopupTrigger()`可以检查鼠标事件是否为平台无关的触发器。SDK 1.3及其以后的版本还提供了与`JPopupMenu`用法相同的等价方法。

当鼠标移出组件时，Swing不再发送弹出触发器事件给组件，弹出式菜单因而不会被唤起。因此，程序员便有机会为不同的底层组件定义各不相同的弹出式菜单，从而增加接口的环境相关性。

属性

JMenuItem类的属性如表14-7所示。弹出式菜单拥有众多的属性，其中的visible属性判断弹出式菜单当前是否显示在屏幕上，setVisible()方法用于显示或隐藏弹出式菜单，不过，当操作对象为自由浮动的弹出式菜单时，调用show()方法会更简便些。location属性提供弹出式菜单当前或曾经在屏幕上的位置，只读的margin属性指定弹出窗口边框和环绕在独立菜单项周围的虚构矩形框之间的间距。

表14-7：JPopupMenu的属性

属性	数据类型	get	is	set	默认值
accessibleContext[o]	AccessibleContext	•			JPopupMenu.accessibleJPopupMenu()
borderPainted	boolean		•	•	true
component	Component	•			
componentAtIndex[i]	Component	•			
invoker	Component	•		•	
label[b]	String	•		•	""
layout[o]	LayoutManager	•		•	GridBagLayout()
lightWeightPopup-Enabled	boolean		•	•	getDefaultLight-WeightPopupEnabled()
location[o]	Point			•	
margin	Insets	•			
popupMenu-Listeners[1.4]	PopupMenu-Listener[]	•			
popupSize	Dimension			•	
selectionModel	SingleSelection-Model	•			DefaultSingle-SelectionModel()
subElements	MenuElement[]	•			
UI[b]	PopupMenuUI	•		•	BasicPopupMenuUI()
UIClassID[o]	String	•			"PopupMenuUI"
visible[b, o]	boolean		•	•	false

[1.4]1.4版以后，[b]绑定，[i]索引，[o]覆盖

请参见表14-4 JMenuItem类的属性

表中的invoker属性是操控弹出式菜单的组件的引用。borderPainted属性指示是

否要绘制弹出式菜单的边框。label属性给每个弹出式菜单指定一个标签，个别外观风格可以根据外观需要自由设置或忽略此属性。请注意，label属性是String对象而非JLabel对象。componentAtIndex是一个索引属性，它返回指定索引的组件。

lightWeightPopupEnabled属性允许程序员启动或禁止对代表弹出式菜单的轻型组件的潜在使用。当该属性为 true 且弹出式菜单位于顶层组件的绘制范围内时，Swing运用轻型组件；反之，当弹出式菜单超出顶层组件的绘制范围时，Swing运用重型（heavyweight）组件。由于接口运用的重型组件会与轻型弹出式菜单相互干扰，所以上述特点应被禁止。程序员可以为所有使用静态 setDefaultLightWeight-PopupEnabled()方法的弹出式菜单设置其 lightWeightPopupEnabled 属性的默认值。

事件

JPopupMenu对象在下列两种状态下会触发一个PopupMenuEvent事件：一是进行菜单的显示/隐藏切换，二是取消菜单项选择。为了维护 PopupMenuEvent 用户列表，该类包含了标准addPopupMenuListener()方法和removePopupMenuListener()方法。

public void addPopupMenuListener(PopupMenuListener l)
public void removePopupMenuListener(PopupMenuListener l)
在对象事件队列中添加或删除一个 PopupMenuListener 监听器。

如果能在弹出式菜单显示之前得到通知，程序员就把握住了根据应用程序的当前状态调整菜单状态和内容的时机，从而使自己的接口更好用，更具环境相关性。

请注意，弹出式菜单被取消的同时也就不再可见，所以这两个事件都是被潜在地触发。不过，现在的实现中似乎很少触发取消事件本身。如果想知道菜单何时消失，就调用 popupMenuWillBecomeInvisible 处理程序。

构造函数

public JPopupMenu()
public JPopupMenu(String title)
创建一个空的弹出式菜单。第二个构造函数接受一个用做弹出式菜单标题的String。

菜单项

public JMenuItem add(JMenuItem menuItem)

public Component add(Component c)

public JMenuItem add(Action a)

上述方法给弹出式菜单添加各种元件。JMenuItem或JComponent对象的子对象都可以被添加，但后者在MenuElement接口的实现中表现更好。如果程序员指定一个Action对象为参数，那么它的众多属性将派生出一个合适的JMenuItem对象，此对象的文字都将居于图像图标的右侧。菜单项会保持与操作的关联，以便操作的任何更新（如名称、图标、允许状态等）都可以在菜单项上反映出来。最后返回生成的JMenuItem对象，程序员可以改变它的格式。

public JMenuItem insert(Action a, int index)

public Component insert(Component component, int index)

在特定的索引处插入一个指定菜单项。上述方法可以接受JComponent或Action对象。如果要实现MenuElement接口，最好使用JComponent为参数。如果程序员指定一个Action对象为参数，那么它的各种属性将派生出一个合适的JMenuItem对象，此对象的文字都将居于图像图标的右侧。菜单项同样会与操作保持关联。最后返回生成的JMenuItem对象，程序员可以改变它的格式。之前位于指定位置或在指定位置之后的所有菜单项的索引都加1。

public void addSeparator()

上述方法给弹出式菜单添加一条分隔线。此分隔线通常是一条横过弹出式菜单的水平线。请注意，分隔线也和菜单项一样算做菜单索引。分隔线是一个内部类的实例，不是普通的JSeparator对象，它始终是水平的。

显示

public void show(Component invoker, int x, int y)

在指定坐标绘制弹出式菜单。该方法使用了一个调用组件的引用。它的功能与下列调用等效：setInvoker()、setLocation()和setvisible()。

public void setPopupSize(int width, int height)

确定弹出式菜单首选尺寸的方法之一（另一种方法是popupSize属性，使用Dimension）。

其他方法

public int getComponentIndex(Component c)

返回与组件引用`c`相关联的索引。如果没有与组件匹配的索引，该方法返回`-1`。

public static boolean getDefaultLightWeightEnabled

返回`lightWeightPopupEnabled`属性的默认值。

public boolean isPopupTrigger(MouseEvent e)

检查给定的鼠标事件是否应触发一个符合当前外观风格的弹出式菜单，SDK 1.3以后使用。

public static void setDefaultLightWeightPopupEnabled(boolean aFlag)

设置`lightWeightPopupEnabled`属性的默认值，用于控制在弹出式菜单中应该使用轻型组件还是重型组件。

public void setSelected(Component c)

强迫弹出式菜单的模型选择一个具体的菜单项。它将迫使弹出式菜单的单个选择模型产生一个属性变化事件。

public void updateUI()

强迫默认的用户界面管理器进行自我更新，以此来重置代理，显示一个新的`PopupMenuUI`。

菜单元素接口

public void menuSelectionChanged(boolean isIncluded)

public MenuElement[] getSubElements()

public Component getComponent()

public void processMouseEvent(MouseEvent event, MenuElement path[], MenuSelectionManager manager)

public void processKeyEvent(KeyEvent event, MenuElement path[], MenuSelectionManager manager)

实现`MenuElement`接口，本章稍后将介绍。

使用弹出式菜单

下面的程序示范了`JPopupMenu`类的使用方法。本例与图14-9中的例子相似，区别在于本例中的弹出式菜单和其中的每一个菜单项都会发出弹出事件。

```
//  PopupMenuExample.java
//
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;

public class PopupMenuExample extends JPanel {

    public JPopupMenu popup;

    public PopupMenuExample() {
        popup = new JPopupMenu();
        ActionListener menuListener = new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                System.out.println("Popup menu item [" +
                                   event.getActionCommand() + "] was pressed.");
            }
        };
        JMenuItem item;
        popup.add(item = new JMenuItem("Left", new ImageIcon("left.gif")));
        item.setHorizontalTextPosition(JMenuItem.RIGHT);
        item.addActionListener(menuListener);
        popup.add(item = new JMenuItem("Center", new ImageIcon("center.gif")));
        item.setHorizontalTextPosition(JMenuItem.RIGHT);
        item.addActionListener(menuListener);
        popup.add(item = new JMenuItem("Right", new ImageIcon("right.gif")));
        item.setHorizontalTextPosition(JMenuItem.RIGHT);
        item.addActionListener(menuListener);
        popup.add(item = new JMenuItem("Full", new ImageIcon("full.gif")));
        item.setHorizontalTextPosition(JMenuItem.RIGHT);
        item.addActionListener(menuListener);
        popup.addSeparator();
        popup.add(item = new JMenuItem("Settings . . ."));
        item.addActionListener(menuListener);

        popup.setLabel("Justification");
        popup.setBorder(new BevelBorder(BevelBorder.RAISED));
        popup.addPopupMenuListener(new PopupPrintListener());

        addMouseListener(new MousePopupListener());
    }

    // An inner class to check whether mouse events are the pop-up trigger
        class MousePopupListener extends MouseAdapter {
        public void mousePressed(MouseEvent e) { checkPopup(e); }
        public void mouseClicked(MouseEvent e) { checkPopup(e); }
        public void mouseReleased(MouseEvent e) { checkPopup(e); }

        private void checkPopup(MouseEvent e) {
            if (e.isPopupTrigger()) {
```

```
                popup.show(PopupMenuExample.this, e.getX(), e.getY());
            }
        }
    }

    // An inner class to show when pop-up events occur
    class PopupPrintListener implements PopupMenuListener {
        public void popupMenuWillBecomeVisible(PopupMenuEvent e) {
            System.out.println("Popup menu will be visible!");
        }
        public void popupMenuWillBecomeInvisible(PopupMenuEvent e) {
            System.out.println("Popup menu will be invisible!");
        }
        public void popupMenuCanceled(PopupMenuEvent e) {
            System.out.println("Popup menu is hidden!");
        }
    }

    public static void main(String s[]) {
        JFrame frame = new JFrame("Popup Menu Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new PopupMenuExample());
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

`MousePopupListener`方法是上述程序中颇有意思的一个部分。程序中调用private方法`checkPopup()`来检查是否收到弹出式菜单的唤起事件。如果获取到一个有效的触发事件,便在鼠标所在位置显示弹出式菜单。这也是覆盖`processMouseEvent()`的一种替代方法，后者在“弹出式菜单的显示”一节中已经介绍过。

PopupMenuEvent类

这是一个告知监听器目标弹出式菜单即将显示、消失或已经被取消的简单事件。请注意,此事件不能判断到底发生了这几种操作中的哪一种。实现`PopupMenuListener`的对象会定义三个独立的方法,以供弹出式菜单调用,每一种都会表明目标弹出式菜单对象的确切操作。

构造函数

public PopupMenuEvent(Object source)

此构造函数接受一个对象的引用，该对象触发事件。

PopupMenuListener 接口

作为 PopupMenuEvent 对象接收管道的 PopupMenuListener 接口共包含 3 种方法。第一种方法在弹出式菜单被取消时调用，另外两种方法指示弹出式菜单将要显示还是隐藏。此接口必须由希望知道弹出式菜单任何变化情况的监听器对象实现。

方法

public abstract void popupMenuCanceled(PopupMenuEvent e)

当目标弹出式菜单被取消或从屏幕上移走时，调用此方法（在实际中很少调用）。

public abstract void popupMenuWillBecomeInvisible(PopupMenuEvent e)

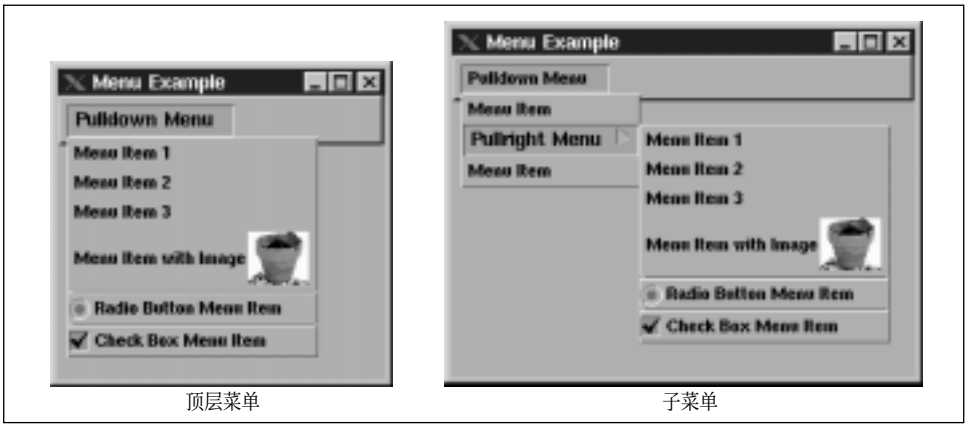
当目标弹出式菜单将要从屏幕上移走时，调用此方法。

public abstract void popupMenuWillBecomeVisible(PopupMenuEvent e)

当目标弹出式菜单将要在屏幕上显示时，调用此方法。程序员应该利用这个机会根据应用程序当前的状态更新菜单内容（或其允许状态）。

JMenu 类

JMenu 类代表连接到 JMenuBar 对象或其他 JMenu 对象上的锚定菜单。直接连接到菜单栏上的菜单被称做顶层（top-level）菜单，子菜单不与菜单栏相连接，而是连接到菜单项上，并以此菜单项为标题。典型的菜单项标题有右箭头标记，表明当用户选中对应菜单项时，菜单项旁边还会弹出一个菜单。参见图 14-11。



顶层菜单　　子菜单

图 14-11：顶层菜单和子菜单

JMenu是一个独特的类。它包含MenuUI代理，却以ButtonModel作为数据模型。之所以如此，是因为可以把菜单被视为菜单项和弹出式菜单两种组件。其中，菜单项是菜单的标题。单击菜单项，它会通知弹出式菜单在它下方或右侧弹出。JMenu实际上是JMenuItem类的子类，因此可以完成菜单标题的实现。这使菜单项事实上成为一种特殊按钮。在某些平台上，程序员可以运用JMenuItem超类的mnemonic属性定义菜单标题和菜单的快捷键。此外，程序员还可以用JMenuItem类的enabled属性在必要时禁止菜单的使用。

程序员可以调用add()和insert()方法在弹出式菜单中添加或插入JMenuItem、Component或Action对象。还可以给菜单添加简单的字符串，JMenu对象自会在内部创建相应的JMenuItem对象。JMenu类给每个菜单项赋予一个整数索引，并依据菜单的布局管理器调整菜单项的顺序。此外，还可以使用addSeparator()方法添加菜单分隔线。

警告：键盘快捷方式对JMenu对象（顶层菜单或子菜单）并不适用，因为快捷方式用于触发真实的程序操作，而不是仅仅显示包含了各种操作命令的菜单。在某些平台上，程序员可以使用setMnemonic()方法设置弹出菜单的快捷方式，但惟一普遍、安全的方法应该是给处理非子菜单的JMenuItems对象指定键盘快捷方式，让它触发程序操作。

将popupMenuVisible属性设置为true可以让子菜单在屏幕上弹出。但要注意的是，当菜单的标题按钮在屏幕上不可见时，弹出式菜单也不会显示。

图14-12显示了JMenu组件的类图。

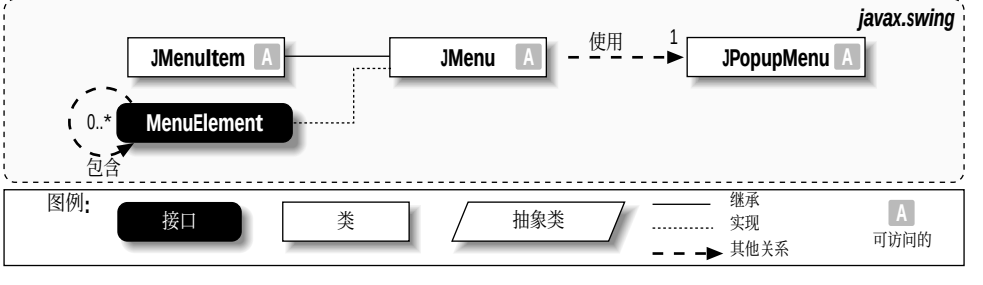


图14-12：JMenu类图

属性

JMenu 的属性如表 14-8 所示。其中，JPopupMenu 代表菜单项列表。popupMenu 属性用于访问底层菜单。popupMenuVisible 属性用于追踪菜单的弹出部分当前是否可见，如上文所说，在标题按钮可见时将它设置为 true 可以显示弹出式菜单。JMenu 还包含 selected 属性，它可以指出用户是否选择了菜单的标题按钮。这两种属性应该相互对应。

表 14-8：JMenu 的属性

属性	数据类型	get	is	set	默认值
accessibleContext[o]	AccessibleContext	•			JMenu.accessibleJMenu()
component	Component	•			
componentOrientation[1.4, o]	ComponentOrientation	•		•	由外观风格决定
delay	int	•		•	0
itemCount	int	•			0
item[i]	JMenuItem	•			null
layout[o]	LayoutManager	•		•	OverlayLayout()
menuComponentCount	int	•			0
menuComponent[i]	Component	•			null
menuComponents	Component[]	•			
menuListeners[1.4]	MenuListener[]	•			
model[o]	ButtonModel	•		•	DefaultButtonModel()
popupMenu	JPopupMenu	•			
popupMenuVisible	boolean		•	•	false
selected	boolean		•	•	false
subElements	MenuElement[]	•			
tearOff[u]	boolean		•		抛出一个 Error
topLevelMenu	boolean		•		
UI[b]	MenuUI			•	由外观风格决定
UIClassID	String	•			"MenuUI"

[1.4]1.4 版以后，[b] 绑定，[i] 索引，[o] 覆盖，[u] 未实现

请参见表 14-4 JMenuItem 类的属性

在上表中，topLevelMenu属性为true表示JMenu直接与菜单栏相连，且不是子菜单。索引属性item允许访问菜单的每一个JMenuItem对象。itemCount属性中包含着当前所有JMenuItem对象的数目。delay属性指定底层菜单收到相应事件后等待显示或消失的时间值（以毫秒为单位）。其值必须为正整数，否则setDelay()将抛出一个IllegalArgumentException异常。

menuComponent属性使item属性具备更好的通用性，它会在指定索引处返回一个Component组件而不是JMenuItem组件。此外，menuComponentCount属性中保存了当前菜单中的菜单项、分隔线和其他组件的数目。menuComponents属性提供给程序员访问菜单项的入口，并返回一个Component对象数组。

componentOrientation属性用于接纳文字方向不是从左到右的非西方的语言。JMenu对此属性做了覆盖，以便将变化适当地传递给它的JPopupMenu代理。

警告：tearOff属性没有被实现，留做Swing将来使用。如果程序员试图使用它，将会导致Error的抛出（其实抛出UnsupportedOperationException更加适合）。由于Error对象被认为是虚拟计算机发生严重故障的标志，所以，使用此属性必定会引起程序崩溃。

构造函数

public JMenu()

public JMenu(Action a)

public JMenu(String s)

public JMenu(String s, boolean b)

初始化默认JMenu对象。可以选择为JMenu指定一个显示字符串、为tearOff属性指定一个布尔值，或是给它绑定一个Action对象。

菜单项

public JMenuItem add(JMenuItem menuItem)

public Component add(Component c)

public void add(String s)

public JMenuItem add(Action a)

上述方法给菜单添加各种元素。JMenuItem和JComponent对象的子对象都可以被添加，但后者在MenuElement接口的实现中表现更好。以String对象为参数将创建一个具有合适标签的菜单项。如果程序员指定一个Action对象为

参数，那么可以利用它的文字和图标属性派生出适当的JMenuItem对象，此对象的文字都将居于图标右侧。此对象将与它的各种操作保持关联，随时更新，以反映其各种属性的改变。最后返回生成的JMenuItem对象，其格式可以改变。

public void 468addSeparator()

上述方法给菜单添加一条分隔线。此分隔线通常是一条横跨弹出式菜单的水平线。

public void insert(String s, int index)

public JMenuItem insert(JMenuItem mi, int index)

public JMenuItem insert(Action a, int index)

在特定的索引处插入一个指定菜单项。索引必须为正整数，否则，该方法将抛出IllegalArgumentException异常。上述方法可以接受JMenuItem、String或Action对象。以String对象为参数将创建一个具有合适标签的菜单项。如果程序员指定一个Action对象为参数，那么可以利用它的文字和图标属性派生出适当的JMenuItem对象，此对象的文字都将居于图标右侧。菜单同样会与操作保持关联。最后返回生成的JMenuItem对象，其格式可以改变。之前位于指定位置或在指定位置之后的所有菜单项的索引都加1。

public void insertSeparator(int index)

在整型的索引参数指定的位置插入一条水平分隔线。索引必须为正整数，否则，该方法将抛出IllegalArgumentException异常。之前位于指定位置或在指定位置之后的所有菜单项的索引都加1。

public void remove(JMenuItem item)

public void remove(int index)

删除与参数JMenuItem相匹配的菜单项，或是当前位于指定整数索引处的菜单项。如果没有与参数匹配的菜单项或不存在该索引位置，菜单将不做改变。如果有菜单项被删除，则位于被删菜单项后的所有菜单项的索引都减1。

public void removeAll()

删除全部菜单项。

其他方法

public void updateUI()

强迫默认的用户界面管理器进行自我更新，以此来重置代理，显示一个新的PopupMenuUI。

public void setMenuLocation(int x, int y)

设置菜单显示时的自定义的位置。

public boolean isMenuComponent(Component c)

确定组件 c 是否位于菜单中。此方法还会搜索所有子菜单。

public String paramString()

返回一个显示菜单属性状态当前的 String 对象（用于程序调试）。

事件

当用户选择或取消选定菜单的标题按钮时，JMenu对象会触发一个MenuEvent事件。JMenu对象包含了标准addChangeListener()方法和removeChangeListener()方法，来维护 MenuEvent 用户列表，

public void addMenuListener(MenuListener listener)

public void removeMenuListener(MenuListener listener)

从接收菜单事件的监听器列表中添加或删除一个 MenuListener。

MenuElement 接口

public void menuSelectionChanged(boolean isIncluded)

public MenuElement[] getSubElements()

public Component getComponent()

public void processKeyEvent(KeyEvent event, MenuElement path[],
MenuSelectionManager manager)

实现 MenuElement 接口，本章稍后将介绍。

使用菜单

下面的程序示范了JMenu类的使用方法。程序使用Swing的Action类处理菜单事件（本章稍后还将在工具栏中使用操作对象）。

```
//  MenuExample.java
//
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
```

```
public class MenuExample extends JPanel {

    public JTextPane pane;
    public JMenuBar menuBar;

    public MenuExample() {
        menuBar = new JMenuBar();
        JMenu formatMenu = new JMenu("Justify");
        formatMenu.setMnemonic('J');

        MenuAction leftJustifyAction = new MenuAction("Left",
                                       new ImageIcon("left.gif"));
        MenuAction rightJustifyAction = new MenuAction("Right",
                                        new ImageIcon("right.gif"));
        MenuAction centerJustifyAction = new MenuAction("Center",
                                         new ImageIcon("center.gif"));
        MenuAction fullJustifyAction = new MenuAction("Full",
                                       new ImageIcon("full.gif"));

        JMenuItem item;
        item = formatMenu.add(leftJustifyAction);
        item.setMnemonic('L');
        item = formatMenu.add(rightJustifyAction);
        item.setMnemonic('R');
        item = formatMenu.add(centerJustifyAction);
        item.setMnemonic('C');
        item = formatMenu.add(fullJustifyAction);
        item.setMnemonic('F');

        menuBar.add(formatMenu);
        menuBar.setBorder(new BevelBorder(BevelBorder.RAISED));

    }

    class MenuAction extends AbstractAction {

        public MenuAction(String text, Icon icon) {
            super(text,icon);
        }

        public void actionPerformed(ActionEvent e) {
            try { pane.getStyledDocument().insertString(0 ,
                  "Action ["+e.getActionCommand()+"] performed!\n", null);
            } catch (Exception ex) { ex.printStackTrace(); }
        }
    }

    public static void main(String s[]) {

        MenuExample example = new MenuExample();
        example.pane = new JTextPane();
        example.pane.setPreferredSize(new Dimension(250, 250));
        example.pane.setBorder(new BevelBorder(BevelBorder.LOWERED));
```

```
            JFrame frame = new JFrame("Menu Example");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setJMenuBar(example.menuBar);
            frame.getContentPane().add(example.pane, BorderLayout.CENTER);
            frame.pack();
            frame.setVisible(true);
        }
    }
```

程序中的 `Action` 对象都是内部类 `MenuActions` 的实例。我们每次为菜单添加 `Action` 对象，菜单都会创建相应的 `JMenuItem`（图像左对齐），并把它返回给程序。这样一来，我们就能以任何需要的方式处理生成的菜单项，在本例中，我们为每个菜单项添加了一个助记符。本程序可以在所有支持助记符的平台上运行。在设计多平台程序时，助记符不应成为用户界面的关键部分，事实上，除非程序员确信它能得到平台的支持，否则应完全摒弃助记符的设置。

结果程序生成一个仅有一个菜单的菜单栏，如图14-13所示。该菜单包含4个菜单项，其外观与前面的弹出式菜单相似。当用户单击菜单项时，Swing 产生一个 `ActionEvent`，由 `MenuAction` 类的 `actionPerformed()` 方法负责处理。如上例中一样，这会显示菜单项的名称。为了体现程序的多样性，我们使用 `JTextPane` 对象显示菜单选择结果，而没有使用系统输出。请参见第十九章和第二十二章中对于 `JTextPane` 的详细介绍。

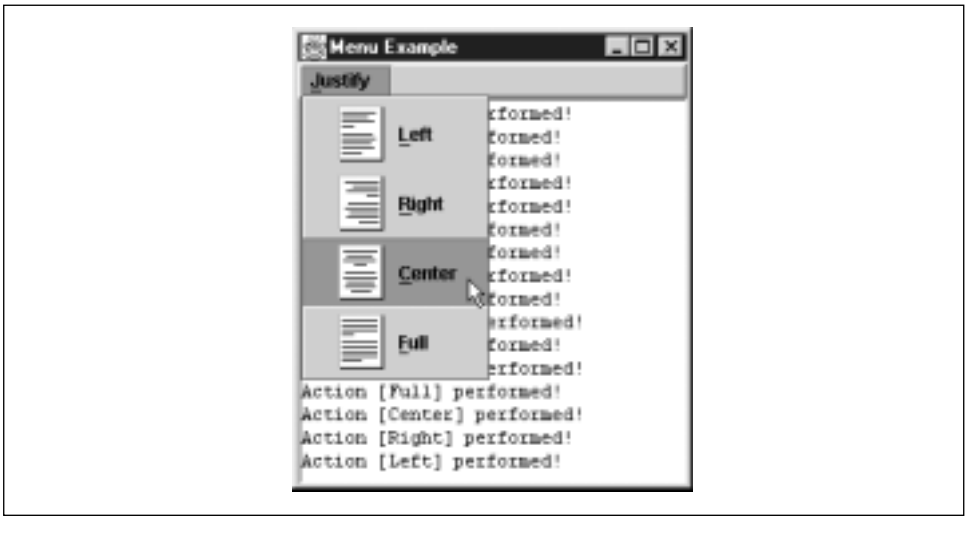


图 14-13：一组使用图标和助记符的菜单项

MenuEvent 类

这是一个告知监听器目标弹出式菜单已经被唤起、选择或取消的简单事件。请注意，此事件不能判断到底发生了这几种操作中的哪一种。监听器定义了3种独立的方法，用于传递菜单事件，每一个方法都能精确判断发生了什么操作。

构造函数

public MenuEvent(Object source)

此构造函数接受一个对象的引用，该对象触发事件。

MenuListener 接口

作为`MenuEvents`接收管道的`MenuListener`接口共指定了3种方法。第一种方法在菜单被取消时调用，另外两种方法在菜单的标题按钮被选定或取消选定时调用。此接口必须由希望知道菜单对象变化情况的监听器对象实现。

方法

public abstract void menuCanceled(MenuEvent e)

当菜单被取消或从屏幕上移走时调用此方法。

public abstract void menuDeselected(MenuEvent e)

当目标菜单的标题按钮被取消选定时调用此方法。

public abstract void menuSelected(MenuEvent e)

当目标菜单的标题按钮被选中时调用此方法。

可选菜单项

迄今为止，本章内容已经涵盖了与动作相关联的产生简单的、面向文字的标签的传统菜单项。但这并不是用户惯用的惟一菜单类型。除此以外，Swing还提供了两种可选菜单项：复选框菜单项和单选按钮菜单项。

JCheckBoxMenuItem 类

复选框菜单项由`JCheckBoxMenuItem`类表示。读者大概已经猜到，这个对象的使用方法类似于`JCheckBox`对象。单击复选框菜单项，用户可以标注或撤消UI定义的选

择标记，此标记通常位于菜单项标签的左侧。相邻的JCheckBoxMenuItem对象之间不会相互排斥，因此用户大可放心检验任一菜单项，不必担心影响到其他项目的状态。图 14-14 显示了 JCheckBoxMenuItem 组件的类图。

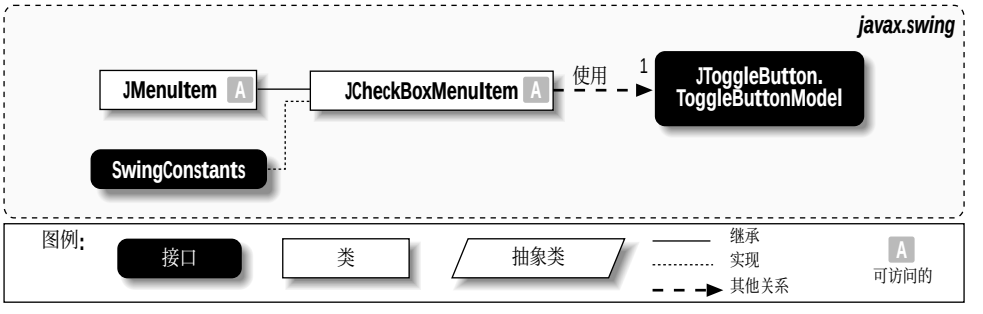


图 14-14：JCheckBoxMenuItem 类图

属性

表 14-9 列出了 JCheckBoxMenuItem 的各个属性。JCheckBoxMenuItem 继承了 JMenuItem模型（ButtonModel）和它的存取方式，还包含两个额外的组件属性。其中，state属性在菜单项被选中时为true，没有选中时为false。selectedObjects 属性存储一个只包含一个元素的 Object 数组，如果菜单项当前处于选中状态，该 Object数组就由菜单项的文字组成；否则，getSelectedObjects()将返回null。getSelectedObjects()方法用于增强 AWT 的 ItemSelectable接口的兼容性。

表 14-9：JCheckBoxMenuItem 的属性

属性	数据类型	get	is	set	默认值
accessibleContext[o]	AccessibleContext	•			JCheckBoxMenuItem.AccessibleJCheckBoxMenuItem()
selectedObjects[o]	Object[]	•			
state	boolean	•		•	false
UI[b]	CheckBoxMenuItemUI	•		•	由外观风格决定
UIClassID[o]	String	•			"CheckBoxMenuItem"

[b] 绑定，[o] 覆盖

请参见表 14-4 JMenuItem 类的属性

构造函数

public JCheckBoxMenuItem()

public JCheckBoxMenuItem(Action action)

public JCheckBoxMenuItem(Icon icon)

public JCheckBoxMenuItem(String text)

public JCheckBoxMenuItem(String text, Icon icon)

public JCheckBoxMenuItem(String text, boolean checked)

public JCheckBoxMenuItem(String text, Icon icon, boolean checked)

上述构造函数使用指定操作（1.3 版以后）、图标或字符串初始化 `JCheckBoxMenuItem`。参数中的布尔值初始化 `state` 属性，确定菜单项的初始状态是否被勾选。

其他方法

public void updateUI()

强迫当前的 UI 管理器重置并重画组件代理，从而更新组件的外观风格。

使用复选框菜单项

下面的程序使用了 `JCheckBoxMenuItem` 类。它与 `JMenu` 的例子相似，只是它的每个菜单项都已被勾选。目前，我们还没有执行任何使各个菜单项互斥的操作，在后面的示例将有所涉及。但是，我们已经对程序代码做出修改，以便使用更方便的键盘快捷方式代替助记符。请注意，我们使用 M（middle）做 Center 选项的快捷方式，因为 C 通常保留给 Copy（复制）命令。图 14-15 显示了程序的结果。

```
// CheckBoxMenuItemExample.java
//
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class CheckBoxMenuItemExample extends JPanel {
    public JTextPane pane;
    public JMenuBar menuBar;
    public JToolBar toolBar;

    public CheckBoxMenuItemExample() {
        menuBar = new JMenuBar();
        JMenu justifyMenu = new JMenu("Justify");
        ActionListener actionPrinter = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
                try { pane.getStyledDocument().insertString(0 ,
                      "Action ["+e.getActionCommand()+"] performed!\n", null);
                } catch (Exception ex) { ex.printStackTrace(); }
            }
        };
        JCheckBoxMenuItem leftJustify = new
               JCheckBoxMenuItem("Left", new ImageIcon("left.gif"));
        leftJustify.setHorizontalTextPosition(JMenuItem.RIGHT);
        leftJustify.setAccelerator(KeyStroke.getKeyStroke('L',
                     Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        leftJustify.addActionListener(actionPrinter);
        JCheckBoxMenuItem rightJustify = new
               JCheckBoxMenuItem("Right", new ImageIcon("right.gif"));
        rightJustify.setHorizontalTextPosition(JMenuItem.RIGHT);
        rightJustify.setAccelerator(KeyStroke.getKeyStroke('R',
                       Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        rightJustify.addActionListener(actionPrinter);
        JCheckBoxMenuItem centerJustify = new
               JCheckBoxMenuItem("Center", new ImageIcon("center.gif"));
        centerJustify.setHorizontalTextPosition(JMenuItem.RIGHT);
        centerJustify.setAccelerator(KeyStroke.getKeyStroke('M',
                       Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        centerJustify.addActionListener(actionPrinter);
        JCheckBoxMenuItem fullJustify = new
               JCheckBoxMenuItem("Full", new ImageIcon("full.gif"));
        fullJustify.setHorizontalTextPosition(JMenuItem.RIGHT);
        fullJustify.setAccelerator(KeyStroke.getKeyStroke('F',
                       Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        fullJustify.addActionListener(actionPrinter);

        justifyMenu.add(leftJustify);
        justifyMenu.add(rightJustify);
        justifyMenu.add(centerJustify);
        justifyMenu.add(fullJustify);

        menuBar.add(justifyMenu);
        menuBar.setBorder(new BevelBorder(BevelBorder.RAISED));

    }

    public static void main(String s[]) {
        CheckBoxMenuItemExample example = new CheckBoxMenuItemExample();
        example.pane = new JTextPane();
        example.pane.setPreferredSize(new Dimension(250, 250));
        example.pane.setBorder(new BevelBorder(BevelBorder.LOWERED));

        JFrame frame = new JFrame("Menu Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setJMenuBar(example.menuBar);
        frame.getContentPane().add(example.pane, BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
    }
}
```

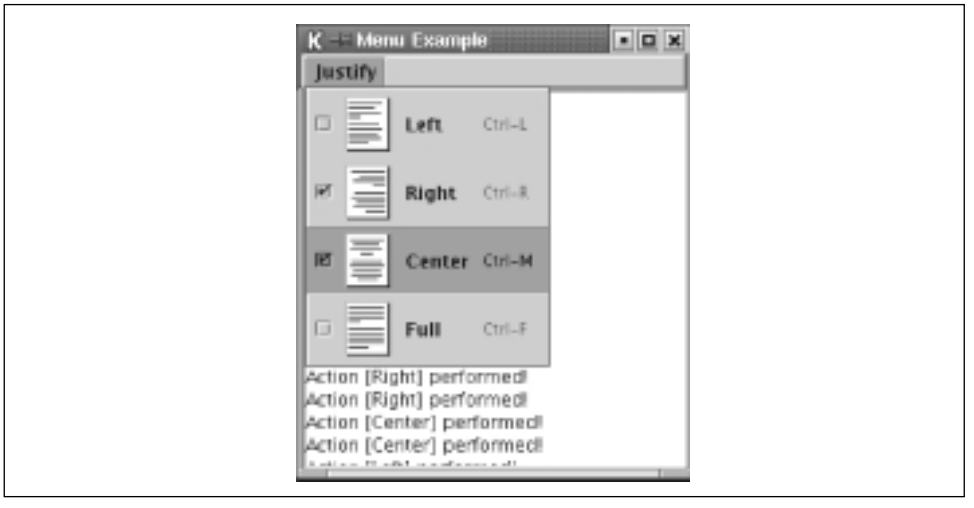


图 14-15：复选框菜单项

JRadioButtonMenuItem 类

Swing 用 `JRadioButtonMenuItem` 类实现单选按钮菜单项。如你所料，它享有 `JRadioButton`类的特征，并代表一组互斥选择。某些外观风格通过选项左侧的圆形按钮来从视觉上表示这种排他性。

注意：即使外观风格在视觉上对复选框和单选按钮进行了区分，用户可能仍然觉得这些区别太细微、不熟悉。所以，使用分隔线等视觉符号暗示菜单内的互斥逻辑分组不失为上佳之选。

单选按钮菜单项不会如程序员所愿自行相互排斥，需要使用`ButtonGroup`对象限制用户只选择单一项目。图 14-16 显示了 `JRadioButtonMenuItem`组件的类图。

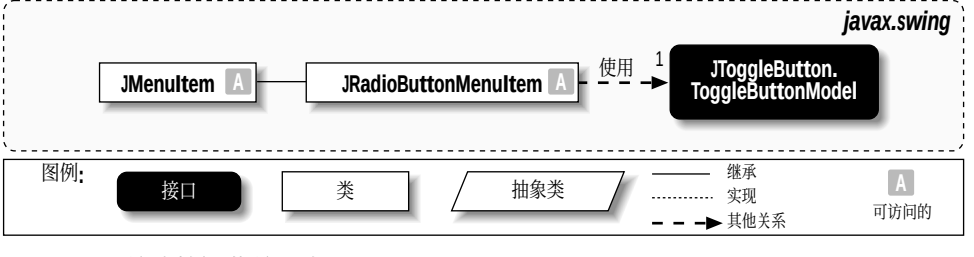


图 14-16：单选按钮菜单项类图

属性

表 14-10 列出了 JRadioButtonMenuItem 的各个属性。与 JCheckBoxMenuItem 不同，JRadioButtonMenuItem 没有指示菜单项当前选择状态的 state 属性。相反，程序员通常在连接ButtonGroup对象时使用这个类，ButtonGroup中包含提取正确对象的 getSelected()方法。

表 14-10：JRadioButtonMenuItem 的属性

属性	数据类型	get	is	set	默认值
accessibleContext[o]	AccessibleContext	•			JRadioButtonMenuItem.AccessibleJRadio-ButtonMenuItem()
UI[b]	RadioButtonMenu-ItemUI	•		•	由外观风格决定
UIClassID[o]	String	•			"RadioButtonMenuItem"

[b] 绑定，[o] 覆盖

请参见表 14-4 JMenuItem 类的属性

构造函数

public JRadioButtonMenuItem()

public JRadioButtonMenuItem(Action action)

public JRadioButtonMenuItem(Icon icon)

public JRadioButtonMenuItem(String text)

public JRadioButtonMenuItem(String text, Icon icon)

上述构造函数使用指定操作（1.3 版以后）、图标或字符串初始化 JRadio-ButtonMenuItem。

其他方法

public void updateUI()

强迫当前的 UI 管理器重置并重画组件代理，从而更新组件的外观风格。

强制互斥

下列程序示范了实现单选按钮菜单项互斥特性的方法。

```
//  RadioButtonMenuItemExample.java
//
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class RadioButtonMenuItemExample extends JPanel {

    public JTextPane pane;
    public JMenuBar menuBar;
    public JToolBar toolBar;

    public RadioButtonMenuItemExample() {
        menuBar = new JMenuBar();
        JMenu justifyMenu = new JMenu("Justify");
        ActionListener actionPrinter = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try { pane.getStyledDocument().insertString(0 ,
                      "Action ["+e.getActionCommand()+"] performed!\n", null);
             } catch (Exception ex) { ex.printStackTrace(); }
             }
        };
        JRadioButtonMenuItem leftJustify = new
               JRadioButtonMenuItem("Left", new ImageIcon("left.gif"));
        leftJustify.setHorizontalTextPosition(JMenuItem.RIGHT);
        leftJustify.setAccelerator(KeyStroke.getKeyStroke('L',
                        Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        leftJustify.addActionListener(actionPrinter);
        JRadioButtonMenuItem rightJustify = new
               JRadioButtonMenuItem("Right", new ImageIcon("right.gif"));
        rightJustify.setHorizontalTextPosition(JMenuItem.RIGHT);
        rightJustify.setAccelerator(KeyStroke.getKeyStroke('R',
                        Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        rightJustify.addActionListener(actionPrinter);
        JRadioButtonMenuItem centerJustify = new
               JRadioButtonMenuItem("Center", new ImageIcon("center.gif"));
        centerJustify.setHorizontalTextPosition(JMenuItem.RIGHT);
        centerJustify.setAccelerator(KeyStroke.getKeyStroke('M',
                        Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        centerJustify.addActionListener(actionPrinter);
        JRadioButtonMenuItem fullJustify = new
               JRadioButtonMenuItem("Full", new ImageIcon("full.gif"));
        fullJustify.setHorizontalTextPosition(JMenuItem.RIGHT);
        fullJustify.setAccelerator(KeyStroke.getKeyStroke('F',
                       Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
        fullJustify.addActionListener(actionPrinter);

        ButtonGroup group = new ButtonGroup();
        group.add(leftJustify);
        group.add(rightJustify);
        group.add(centerJustify);
```

```
            group.add(fullJustify);

            justifyMenu.add(leftJustify);
            justifyMenu.add(rightJustify);
            justifyMenu.add(centerJustify);
            justifyMenu.add(fullJustify);

            menuBar.add(justifyMenu);
            menuBar.setBorder(new BevelBorder(BevelBorder.RAISED));
        }

        public static void main(String s[]) {

            RadioButtonMenuItemExample example = new
                                    RadioButtonMenuItemExample();
            example.pane = new JTextPane();
            example.pane.setPreferredSize(new Dimension(250, 250));
            example.pane.setBorder(new BevelBorder(BevelBorder.LOWERED));

            JFrame frame = new JFrame("Menu Example");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setJMenuBar(example.menuBar);
            frame.getContentPane().add(example.pane, BorderLayout.CENTER);
            frame.pack();
            frame.setVisible(true);
        }
    }
```

图 14-17 显示了程序的结果。我们使用 `ButtonGroup` 对象创建了互斥的 `JRadioButtonMenuItem` 对象。对任何菜单项的选择将导致对其他菜单项的选择被取消。本例子中互斥的正文对齐方式显示了实现真实的对齐方式菜单的方法。

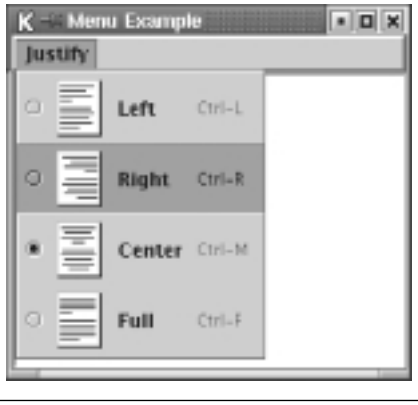


图 14-17：单选按钮菜单项示例

JSeparator 类

读者也许已经注意到，JMenu 和 JPopupMenu 都包含给菜单添加分隔线的 addSeparator()方法。通过这种方法，每个类都可以实例化 JSeparator对象，并让它定位于菜单内。但是，由于JSeparator以菜单外的组件形式存在，并且继承自 JComponent，所以，它也可以像其他任何Swing组件一样位于容器内部。JSeparator 是一个让菜单项的不同逻辑组相互隔开的简单组件，在某些外观风格中，它表现为一条横过整个菜单宽度的水平线；在另一些情况下，它被隐藏起来，只在菜单元素之间增加一点间距而已。JSeparator 没有模型，只有一个代理。

属性

表 14-11 列出了 JSeparator 的各个属性。

表 14-11：JSeparator 的属性

属性	数据类型	get	is	set	默认值
accessibleContext[o]	AccessibleContext	•			JSeparator.accessibleJSeparator()
orientation	int	•		•	SwingConstants.HORIZONTAL
UI[b]	SeparatorUI	•		•	由外观风格决定
UIClassID[o]	String	•			"SeparatorUI"

[b] 绑定，[o] 覆盖

请参见表 3-6 的 JComponent 类属性

构造函数

JSeparator()

JSeparator(int orientation)

创建一条分隔线。默认情况下，分隔线是水平横线；如果程序员要指定它的方向，可以使用常量 SwingConstants.HORIZONTAL 或SwingConstants.VERTICAL。

其他方法

public void updateUI()

强迫当前的 UI 管理器重置并重画组件代理，从而更新组件的外观风格。

在菜单外使用分隔线

我们已经看到怎样使用菜单分隔线高亮显示一组菜单项。事实上，作为一种组件，分隔线还能完成多种功能。下面的程序是在一连串按钮之间添加了一条分隔线：

```
// SeparatorExample.java
//
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class SeparatorExample extends JPanel {

    public SeparatorExample() {
        super(true);

        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        Box box1 = new Box(BoxLayout.X_AXIS);
        Box box2 = new Box(BoxLayout.X_AXIS);
        Box box3 = new Box(BoxLayout.X_AXIS);

        box1.add(new JButton("Press Me"));
        box1.add(new JButton("No Me!"));
        box1.add(new JButton("Ignore Them!"));
        box2.add(new JSeparator());
        box3.add(new JButton("I'm the Button!"));
        box3.add(new JButton("It's me!"));
        box3.add(new JButton("Go Away!"));

        add(box1);
        add(box2);
        add(box3);
    }

    public static void main(String s[]) {

        SeparatorExample example = new SeparatorExample();

        JFrame frame = new JFrame("Separator Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(example);
        frame.pack();
        frame.setVisible(true);
    }
}
```

这段代码生成的结果如图14-18所示。请注意，平台上的分隔线都应该是可见的，但如果使用不太正规的调用方法将使分隔线很难或几乎不可能被察觉。尽管本例中的分隔线已经被直观表示，但仍然非常不明显。

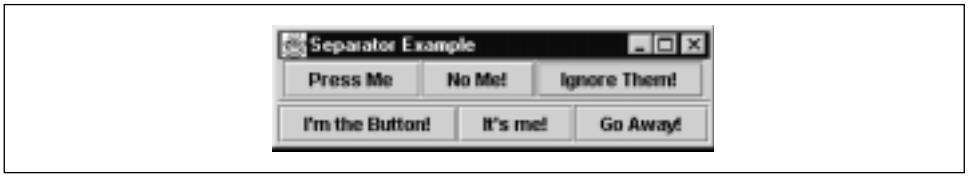


图 14-18：两组按钮之间的独立分隔线

MenuElement 接口

正如前例所示，Swing菜单的一大优点是不对菜单项强制使用文字。不过，这也不是说必须用图标。事实上，你只需稍下功夫，就可以将任何创建或继承的Java组件用到菜单项中。但有一点务必注意：新的菜单项必须实现 `MenuElement` 接口。Swing在 `MenuElement` 接口中声明了5种方法，当各种操作发生时，Swing的内部对象 `MenuSelectionManager` 将调用这些方法。

为什么要这么做？让我们以Edit（编辑）菜单中的传统菜单项Paste（粘贴）为例来分析。当用户唤起Edit菜单，鼠标掠过Paste的菜单项时，菜单项一般会通过颜色的变化使自己突出显示，从而提示用户释放鼠标（或单击鼠标，取决于外观风格）选择Paste选项。在鼠标离开菜单项后，菜单项恢复其标准色。但是，如果我们想让文字变粗而不是突出显示，该怎么办？如果我们想在鼠标经过某个菜单项时更换它的图标图像，又该怎么办？通过调用接口中的方法，菜单允许菜单项定义自己的独特特性。

方法

public void processMouseEvent(MouseEvent event,MenuElement path[],
MenuSelectionManager manager)

此方法处理鼠标触发的事件。除了 `MouseEvent` 外，其参数还包括被选中菜单元素的当前路径和当前菜单选择管理器的引用。程序员可以使用此方法执行任何认为必要的操作。

public void processKeyEvent(KeyEvent event, MenuElement path[],
MenuSelectionManager manager)

此方法处理按键触发的事件。除了 `KeyEvent` 外，其参数还包括被选中菜单元素的当前路径和当前菜单选择管理器的引用。程序员可以使用此方法执行任何认为必要的操作。

public void menuSelectionChanged(boolean isIncluded)

当菜单元素被添加进当前目标菜单或从目标菜单上被删除时调用此方法。

public MenuElement[] getSubElements()

返回目标 `MenuElement` 的一组子元素。它适用于带有子菜单的菜单事件。

public Component getComponent()

返回一个负责绘制菜单项的组件的引用。

将任意组件转换成菜单元件

将任意的 Swing 组件变成菜单元素并把它放到菜单中，相对较容易。下面的程序将一个 `JSlider` 对象放到弹出式菜单中，并用它暗中控制底层组件。

```
//  MenuElementExample.java
//
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;

public class MenuElementExample extends JPanel {

    public JPopupMenu popup;
    SliderMenuItem slider;
    int theValue = 0;

    public MenuElementExample() {

        popup = new JPopupMenu();
        slider = new SliderMenuItem();

        popup.add(slider);
        popup.add(new JSeparator());

        JMenuItem ticks = new JCheckBoxMenuItem("Slider Tick Marks");
        ticks.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                slider.setPaintTicks(!slider.getPaintTicks());
            }
        });
        JMenuItem labels = new JCheckBoxMenuItem("Slider Labels");
        labels.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                slider.setPaintLabels(!slider.getPaintLabels());
            }
```

```
        });
        popup.add(ticks);
        popup.add(labels);
        popup.addPopupMenuListener(new PopupPrintListener());

        addMouseListener(new MousePopupListener());
    }

    // Inner class to check whether mouse events are the pop-up trigger
    class MousePopupListener extends MouseAdapter {
        public void mousePressed(MouseEvent e) { checkPopup(e); }
        public void mouseClicked(MouseEvent e) { checkPopup(e); }
        public void mouseReleased(MouseEvent e) { checkPopup(e); }

        private void checkPopup(MouseEvent e) {
            if (e.isPopupTrigger()) {
                popup.show(MenuElementExample.this, e.getX(), e.getY());
            }
        }
    }

    // Inner class to print information in response to pop-up events
    class PopupPrintListener implements PopupMenuListener {
        public void popupMenuWillBecomeVisible(PopupMenuEvent e) { }

        public void popupMenuWillBecomeInvisible(PopupMenuEvent e) {
            theValue = slider.getValue();
            System.out.println("The value is now " + theValue);
        }

        public void popupMenuCanceled(PopupMenuEvent e) {
            System.out.println("Popup menu is hidden!");
        }
    }

    public static void main(String s[]) {
        JFrame frame = new JFrame("Menu Element Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new MenuElementExample());
        frame.setSize(300, 300);
        frame.setVisible(true);
    }

    // Inner class that defines our special slider menu item
    class SliderMenuItem extends JSlider implements MenuElement {

        public SliderMenuItem() {
            setBorder(new CompoundBorder(new TitledBorder("Control"),
                                         new EmptyBorder(10, 10, 10, 10)));

            setMajorTickSpacing(20);
            setMinorTickSpacing(10);
        }
```

```
            public void processMouseEvent(MouseEvent e, MenuElement path[],
                                          MenuSelectionManager manager) {}

            public void processKeyEvent(KeyEvent e, MenuElement path[],
                                        MenuSelectionManager manager) {}

            public void menuSelectionChanged(boolean isIncluded) {}

            public MenuElement[] getSubElements() {return new MenuElement[0];}

            public Component getComponent() {return this;}
        }
    }
```

与前面举过的PopupMenuExample弹出式菜单例子一样，我们通过实现MouseListener并检查鼠标的进入事件来确定是否显示弹出式菜单。内部类SliderMenuItem实现MenuElement接口，它是本例的焦点所在。在本例中，它的用法相当简单。菜单滑尺没有子元素、不存在选择的概念、也不需要鼠标或键盘事件完成任何特殊操作。

本例实现的界面如图14-19所示。其中包含一个JSlider对象、一条分隔线和两个JCheckBoxMenuItem对象，共同控制滑尺状态。滑尺外还有一个带标题的边框。当用户调整滑尺并消除弹出式菜单时，程序在标准输出设备上显示滑尺的当前值。其实，富有想象力的程序员可以用弹出式菜单完成各种功能。当然，如果所得结果太离谱，程序员也应当仔细考虑它是否会对用户造成困扰。

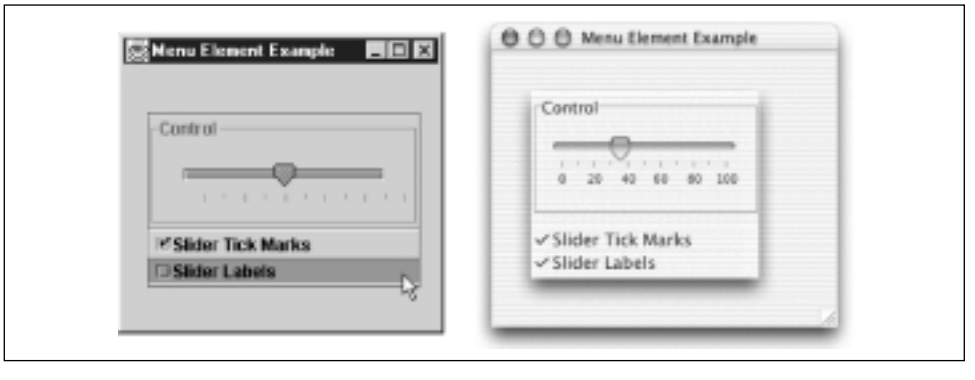


图14-19：伪装成弹出式菜单元素的JSlider（两种外观风格）

工具栏

另一个访问应用程序常用特性的方法是工具栏。与菜单相比，工具栏更适合使用图像

来表示命令。因其总是显示于屏幕上（不像菜单只有在激活时才弹出），所以便能提供一个“控制盘”显示应用程序的当前状态。但是，工具栏占据的空间也较菜单栏为大，故让用户自行决定是否让工具栏可见是一种不错的方法。

工具栏能够将自身从所在的框架中“分裂”出来，把它包含的组件嵌入可移动的独立窗口中。这一特点使用户可以在屏幕上自由拖动工具栏。此外，它还可以“停放”在布局管理器支持的任何位置。

JToolBar 类

与菜单栏一样，`JToolBar`类是各种组件的容器。按钮、组合框，甚至菜单等任何组件都可以添加进工具栏中。如同菜单一样，工具栏与`Action`对象配套使用时操作最简便。

当某个组件被添加到工具栏里后，它即被赋予一个整数索引，以确定其从左至右的显示顺序。虽然工具栏组件的类型没有任何限制，但是，当组件高度相等时显示效果通常最佳。请注意，工具栏的默认边框由外观风格决定。如果不喜欢该默认设置，可以用`setBorder()`方法创建新的设置，覆盖默认边框。或者把`borderPainted`属性设为`false`，使`border`的绘制无效。

`JToolBar`的分隔线是在工具栏中插入空白区。`addSeparator()`方法可以访问该分隔线。分隔线在程序员想增大相关的工具栏组件组之间的间距时十分有用，它实际上是一个内部类。请务必分清分隔线和`JSeparator`类。

图 14-20 显示了`JToolBar`组件的类图。

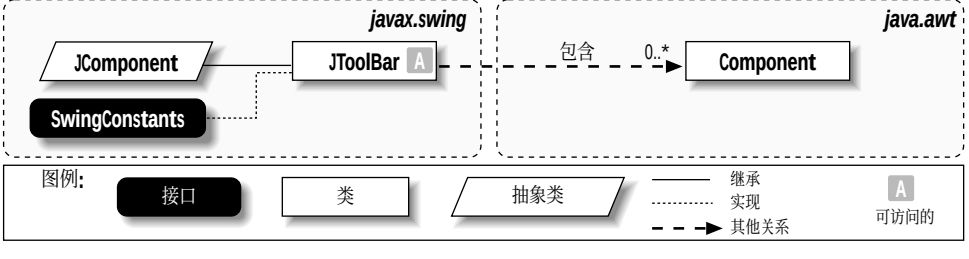


图 14-20：JToolBar 类图

浮动工具栏

工具栏虽然很容易在 Swing 容器中定位，却不必一定位于容器中。相反，程序员可以在光标位于工具栏的空白区域时（即鼠标不在任何组件之上时），按下鼠标不放并拖动，从而“浮动”工具栏。工具栏被拖到一个可移动的子窗口中，可位于视图区的任何位置。这样一来，工具栏便再次附着于框架内的特定位置或热区。如果拖动工具栏的鼠标在热区上释放，工具栏即被锚定回容器中。图 14-21 显示了一个浮动工具栏。

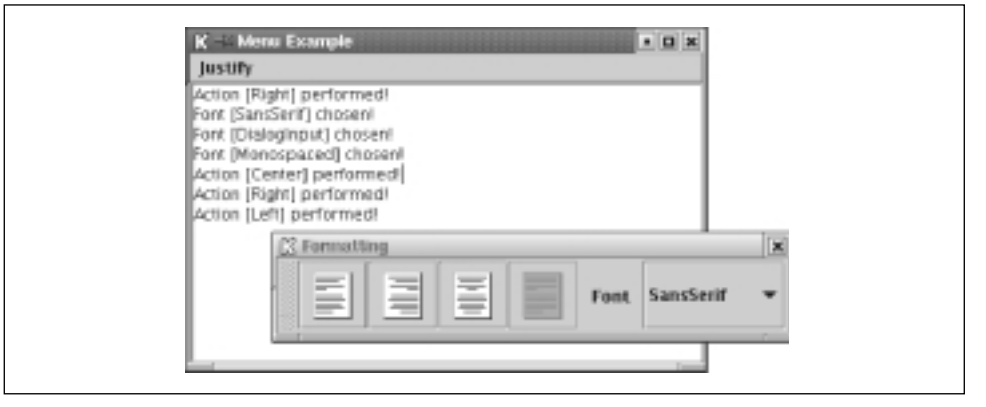


图 14-21：浮动工具栏

把工具栏置于`BorderLayout`支持的容器中是最好的做法。如果想让工具栏浮动，可以将它沿容器的任何一条边放置，让其余的边开放。这样一来，工具栏就可在被拖动时定义锚点，确保最后生成的布局清楚得当。

如果你不想让工具栏浮动，可以将 `floatable` 属性重置为 `false`：

```
JToolBar toolBar = new JToolBar();
toolBar.setFloatable(false);
```

属性

`JToolBar` 类的属性如表 14-12 所示。其中，`borderPainted`属性定义是否要绘制工具栏的边框。`JToolBar` 构造函数将组件的布局管理器重置为沿横坐标轴方向的 `BoxLayout`（当`orientation` 属性为 `VERTICAL` 时是沿 Y 轴的 `BoxLayout`），用于放置子类组件。`margin`属性定义工具栏边缘和其组件之间的镶边。`floatable`属性定义工具栏是否可以脱离容器，在独立窗口中“浮动”。带索引的`componentAtIndex`属性可以访问工具栏中的任何组件。`orientation`属性确定工具栏是横向

还是纵向，它的取值为HORIZONTAL或VERTICAL两者之一，它们都是SwingConstants定义的常量。orientation的其他取值都将导致IllegalArgumentException的抛出。最后，SDK 1.4以后添加的rollover属性可以让工具栏的按钮边框只有在鼠标经过时才被绘制，这种界面风格在动态网站中应用广泛。但并不是所有外观风格都认可rollover的true值。

表14-12：JToolBar的属性

属性	数据类型	get	is	set	默认值
accessibleContext	AccessibleContext	•			JToolBar.AccessibleJToolBar()
borderPainted[b]	boolean		•	•	true
componentAtIndex[i]	Component	•			
floatable[b]	boolean		•	•	true
layout[o]	LayoutManager	•		•	BoxLayout(X_AXIS)
margin[b]	Insets	•		•	Insets(0,0,0,0)
orientation[b]	int	•		•	SwingConstants.HORIZONTAL
rollover[1.4, b]	boolean		•	•	false
UI[b]	ToolBarUI	•		•	由外观风格决定
UIClassID[o]	String	•			"ToolBarUI"

[1.4]1.4版以后，[b]绑定，[i]索引，[o]覆盖

请参见表3-6的JComponent类属性

事件

JToolbar的任何绑定属性变化时都会产生一个PropertyChangeEvent事件。

构造函数

public JToolBar()

public JToolBar(int orientation)

public JToolBar(String name)

public JToolBar(String name, int orientation)

创建一个JToolBar对象，可以有选择地提供一个名称，作为工具栏浮动时显示的标题（SDK 1.3以后支持此功能）。标题文字的显示方向默认为水平方向。

如果程序员要指定它的方向，只能使用常量SwingConstants.HORIZONTAL或SwingConstants.VERTICAL。

添加操作

public JButton add(Action a)

给工具栏添加一个Action对象。此方法创建一个简易的JButton对象，操作文字置于图像下面（注1）。它返回JButton对象，允许重置按钮的任何属性。此方法继承自Container。

其他方法

public void updateUI()

强迫当前的UIuimanager重画组件的UI代理，更新组件的外观风格。

public int getComponentIndex(Component c)

返回组件c的整数索引。返回值-1表示没有找到组件。请注意，工具栏中的任何分隔线都占据索引位置。

public void addSeparator()

public void addSeparator(Dimension size)

上述方法给工具栏添加一条分隔线。请务必分清工具栏分隔线和JSeparator，后者是一个独立的Swing组件。此方法创建的工具栏分隔线只是一段增加不同工具栏组件组之间间距的空白。其大小一般取决于工具栏，但是，程序员可以按自己的要求明确指定分隔线尺寸。

创建工具栏

下面的例子为一个JMenu实例添加了一个工具栏，并显示了使用Action对象构造用户界面的强大功能。

为了增加程序的趣味性和复杂性，我们允许用户从组合框中选择字体，以示可以在工具栏中使用其他类型的组件。请注意，组合框和JLabel对象是作为独立组件分别添加的，组合框调用了它自己特有的actionPerformed()方法。

注1： SDK 1.3中，Sun公司不推荐使用这种方法，也没有实际否决它。人们已经认识到并确认没有适当的替代方案。所以当使用它的时候，要时刻关注最新消息。

```
//  ToolBarExample.java
//
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;

public class ToolBarExample extends JPanel {

    public JTextPane pane;
    public JMenuBar menuBar;
    public JToolBar toolBar;
    String fonts[] = {"Serif","SansSerif","Monospaced","Dialog","DialogInput"};

    public ToolBarExample() {
        menuBar = new JMenuBar();

        // Create a set of actions to use in both the menu and toolbar.
        DemoAction leftJustifyAction = new DemoAction("Left",
            new ImageIcon("left.gif"), "Left justify text", 'L');
        DemoAction rightJustifyAction = new DemoAction("Right",
            new ImageIcon("right.gif"), "Right justify text", 'R');
        DemoAction centerJustifyAction = new DemoAction("Center",
            new ImageIcon("center.gif"), "Center justify text", 'M');
        DemoAction fullJustifyAction = new DemoAction("Full",
            new ImageIcon("full.gif"), "Full justify text", 'F');

        JMenu formatMenu = new JMenu("Justify");
        formatMenu.add(leftJustifyAction);
        formatMenu.add(rightJustifyAction);
        formatMenu.add(centerJustifyAction);
        formatMenu.add(fullJustifyAction);
        menuBar.add(formatMenu);

        toolBar = new JToolBar("Formatting");
        toolBar.add(leftJustifyAction);
        toolBar.add(rightJustifyAction);
        toolBar.add(centerJustifyAction);
        toolBar.add(fullJustifyAction);

        toolBar.addSeparator();
        JLabel label = new JLabel("Font");
        toolBar.add(label);

        toolBar.addSeparator();
        JComboBox combo = new JComboBox(fonts);
        combo.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try { pane.getStyledDocument().insertString(0,
                    "Font [" + ((JComboBox)e.getSource()).getSelectedItem() +
                    "] chosen!\n", null);
                } catch (Exception ex) { ex.printStackTrace(); }
```

```
            }
        });
        toolBar.add(combo);

        // Disable one of the Actions.
        fullJustifyAction.setEnabled(false);
    }

    public static void main(String s[]) {

        ToolBarExample example = new ToolBarExample();
        example.pane = new JTextPane();
        example.pane.setPreferredSize(new Dimension(250, 250));
        example.pane.setBorder(new BevelBorder(BevelBorder.LOWERED));
        example.toolBar.setMaximumSize(example.toolBar.getSize());

        JFrame frame = new JFrame("Menu Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setJMenuBar(example.menuBar);
        frame.getContentPane().add(example.toolBar, BorderLayout.NORTH);
        frame.getContentPane().add(example.pane, BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
    }

    class DemoAction extends AbstractAction {

        public DemoAction(String text, Icon icon, String description,
                          char accelerator) {
            super(text, icon);
            putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(accelerator,
                     Toolkit.getDefaultToolkit().getMenuShortcutKeyMask()));
            putValue(SHORT_DESCRIPTION, description);
        }

        public void actionPerformed(ActionEvent e) {
            try { pane.getStyledDocument().insertString(0,
                  "Action [" + getValue(NAME) + "] performed!\n", null);
            } catch (Exception ex) { ex.printStackTrace(); }
        }
    }
}
```

请注意程序的执行效率：通过创建一套代表对齐模式的Action对象，我们仅用一行代码就创建出了相应的菜单入口或工具条按扭。每个Action对象都有适当的标签和（或）图标、快捷方式（用于菜单）和工具提示。将鼠标置于工具栏按钮上，能看到工具提示。还需注意，一旦禁止了Full（两端）对齐操作，其相应的菜单项和工具条按扭也就自动被禁止。当某个操作被禁止时，所有与之关联的组件都被告知属性发生的变化。在这个程序中，应用两端对齐方式的菜单项和工具条按扭呈现灰色，如图14-22所示。

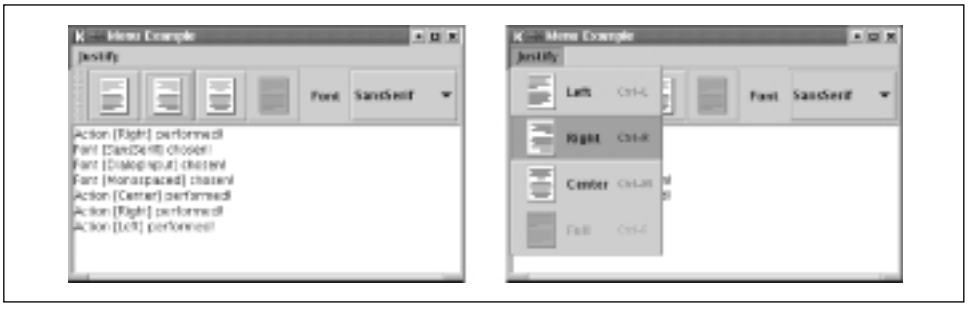


图 14-22：禁止操作自动让工具栏和菜单显示为灰色

要让工具栏浮动，请单击工具栏左边的区域，将它拖到窗口外部。也可以将它拖到框架内的任意边缘处，使之固定在那里。

`JToolBar` 是一个常规的 Swing 组件，因而可以在应用程序中多次使用。当程序员使用了多个 `JToolBar`，而且希望所有工具栏都可浮动时，最好把每个工具栏都置于一个中心对齐的 `BorderLayout` 容器中，让其余三条边空闲。这就保证了工具栏在被拖到一个新边时仍保持各自的位置。