


第19章

高级布局管理器

( 视频讲解：39 分钟)

Swing 还提供了一些高级布局管理器，如箱式布局管理器、卡片布局管理器、网格组布局管理器以及弹簧布局管理器，通过对这些布局管理器的使用，可以设计出更好、更实用的程序界面。在讲解过程中为了便于读者理解，结合了大量的图例，还针对每个知识点进行了举例。

通过阅读本章，您可以：

- » 学会箱式布局管理器的使用方法
- » 学会卡片布局管理器的使用方法
- » 学会网格组布局管理器的使用方法
- » 学会弹簧布局管理器的使用方法
- » 了解弹簧和支柱的使用方法
- » 能够利用弹簧和支柱显示组件的大小：
- » 学会利用各种布局管理器设计程序界面

学
乎
知
学

PDG

19.1 箱式布局管理器

 视频讲解：光盘\TM\lx\19\箱式布局管理器.exe

由 `BoxLayout` 类实现的布局管理器称为箱式（`BoxLayout`）布局管理器，用来管理一组水平或垂直排列的组件，如果是用来管理一组水平排列的组件，则称为水平箱，效果如图 19.1 所示。

如果是用来管理一组垂直排列的组件，则称为垂直箱，效果如图 19.2 所示。



图 19.1 水平箱

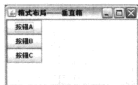


图 19.2 垂直箱

`BoxLayout` 类提供了一个构造方法 `BoxLayout(Container target, int axis)`，其入口参数 `target` 为要采用该布局方式的容器对象；入口参数 `axis` 为要采用的布局方式，如果将其设置为静态常量 `X_AXIS`，表示创建一个水平箱，组件将从左到右排列，设置为静态常量 `Y_AXIS` 则表示创建一个垂直箱，组件将从上到下排列。无论水平箱还是垂直箱，当将窗体调小至不能显示所有组件时，组件仍会排列在一行或一列，如图 19.3 所示，组件的排列顺序依据添加到容器中的先后顺序。

箱式布局在排列组件时，对于水平箱，会试图将所有组件的高度调整到其中最高组件的高度，如果某一组件不能满足这个要求，会根据该组件的垂直调整值（即 `alignmentY` 属性的值）进行调整。具体的调整方案如图 19.4 所示。

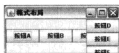


图 19.3 调小窗体效果

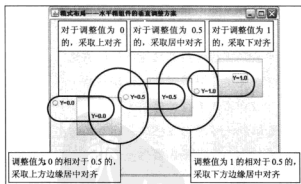


图 19.4 水平箱组件的垂直调整方案

对于垂直箱，会试图将所有组件的宽度调整到其中最宽组件的宽度，如果某一组件不能满足这个要求，会根据该组件的水平调整值（即 `alignmentX` 属性的值）进行调整。具体的调整方案如图 19.5 所示。

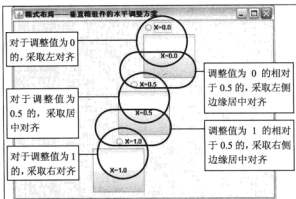


图 19.5 垂直箱组件的水平调整方案

默认情况下箱式布局管理器的组件之间没有间距，如果需要在组件之间设置间距，可以通过使用 Box 类提供的 6 个不可见组件实现，这些组件就是专门用来设置箱式布局的。

Box 类是以 BoxLayout 类为其布局管理器的轻量级容器，可以分别通过静态方法 createHorizontalBox() 和 createVerticalBox() 获得水平或垂直布局管理器的箱式容器。在该类中提供了两种类型的不可见的组件，分别为支柱 (Strut) 类型和胶水 (Glue) 类型，它们具体包含的组件如图 19.6 所示。



图 19.6 Box 类提供的不可见组件

支柱类型的组件具有指定的宽度、高度或大小，通常用来设置组件间的固定间距，例如使用水平支柱设置 3 个按钮间距后的效果如图 19.7 所示。

胶水类型的组件类似于一个弹簧，通常用来将组件平均分布到容器中，例如使用水平胶水设置 3 个按钮间距后的效果如图 19.8 所示。

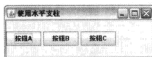


图 19.7 使用水平支柱设置效果



图 19.8 使用水平胶水设置效果

【例 19.1】 使用箱式布局管理器实现组件的右对齐和上对齐，以及控制组件之间的间距。(实例位置：光盘\TM\sl\19\1)

```
Box topicBox = Box.createHorizontalBox(); //创建一个水平箱容器
getContentPane().add(topicBox, BorderLayout.NORTH); //添加到窗体中
topicBox.add(Box.createHorizontalStrut(5)); //添加一个 5 像素宽的水平支柱
JLabel topicLabel = new JLabel("主题："); //创建主题标签
```

topicBox.add(topicLabel);	//添加到水平容器中
topicBox.add(Box.createHorizontalStrut(5));	//添加一个 5 像素宽的水平支柱
TextField topicTextField = new JTextField(30);	//创建文本框
topicBox.add(topicTextField);	//添加到水平容器中
Box box = Box.createVerticalBox();	//创建一个垂直容器
getContentPane().add(box, BorderLayout.CENTER);	//添加到窗体中
box.add(Box.createVerticalStrut(5));	//添加一个 5 像素高的垂直支柱
Box contentBox = Box.createHorizontalBox();	//创建一个水平容器
contentBox.setAlignmentX(1);	//设置组件的水平调整值，靠右侧对齐
box.add(contentBox);	//添加到垂直容器中
contentBox.add(Box.createHorizontalStrut(5));	//添加一个 5 像素宽的水平支柱
JLabel contentLabel = new JLabel("内容：");	//创建一个标签组件
contentLabel.setAlignmentY(0);	//设置组件的垂直调整值，靠上方对齐
contentBox.add(contentLabel);	//添加到水平容器中
contentBox.add(Box.createHorizontalStrut(5));	//添加一个 5 像素宽的水平支柱
JScrollPane scrollPane = new JScrollPane();	//创建一个滚动面板
scrollPane.setAlignmentY(0);	//设置组件的垂直调整值，靠上方对齐
contentBox.add(scrollPane);	//添加到水平容器中
JTextArea contentTextArea = new JTextArea();	//创建一个文本区域
scrollPane.setViewportView(contentTextArea);	//添加到滚动面板中
box.add(Box.createVerticalStrut(5));	//添加一个 5 像素高的垂直支柱
JButton submitButton = new JButton("确定");	//创建一个按钮
submitButton.setAlignmentX(1);	//设置组件的水平调整值，靠右侧对齐
box.add(submitButton);	//添加到垂直容器中

运行结果如图 19.9 所示。

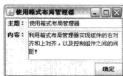


图 19.9 使用箱式布局管理器

窗体中的内容标签和内容文本区实现了上对齐，并且它们所在的水平容器和“确定”按钮实现了靠右侧对齐。

19.2 卡片布局管理器

 视频讲解：光盘\TM\19\卡片布局管理器.exe

由 CardLayout 类实现的布局管理器称为卡片 (CardLayout) 布局管理器，用来操纵其所管理容器中包含的容器或组件。每个直接添加到其所管理容器中的容器或组件为一个卡片，最先被添加的容器或组件被认为是第一个卡片，最后被添加的则为最后一个卡片，初次运行时将显示第一个卡片。

在 CardLayout 类中提供了 5 个用来显示卡片的方法，如表 19.1 所示，具体的使用方法如图 19.10 所示。

表 19.1 CardLayout 类中用来显示卡片的方法

方 法	说 明
first(Container parent)	显示第一个卡片
last(Container parent)	显示最后一个卡片
next(Container parent)	显示当前所显示卡片之后的卡片
previous(Container parent)	显示当前所显示卡片之前的卡片
show(Container parent, String name)	显示指定标签的卡片

**说明**

表 19.1 中的 5 个方法均需要传入一个 Container 型的对象, 该对象为布局管理器对象所管理的容器对象。

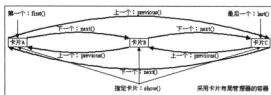


图 19.10 显示采用卡片布局管理器的容器中的卡片

**注意**

当利用 next() 或 previous() 方法显示卡片时, 将循环显示所有卡片。例如, 如果当前显示的是最后一个卡片, 再调用 next() 方法将显示第一个卡片; 如果当前显示的是第一个卡片, 再调用 previous() 方法将显示最后一个卡片。

如果需要使用方法 show(Container parent, String name) 显示卡片, 则需要利用方法 add(Component comp, Object constraints) 向其所管理的容器中添加组件, 其中入口参数 parent 为卡片对象, name 为卡片的标签。

【例 19.2】 使用卡片布局管理器的方法, 并利用 CardLayout 类中用来显示卡片的方法实现各种显示卡片的按钮。(实例位置: 光盘\TM\sl\19\2)

```
public class ExampleFrame_02 extends JFrame {
    private JPanel cardPanel; //采用卡片布局管理器的面板对象
    private CardLayout cardLayout; //卡片布局管理器对象
    public static void main(String args[]) {
        ExampleFrame_02 frame = new ExampleFrame_02();
        frame.setVisible(true);
    }
    public ExampleFrame_02() {
        super();
        setTitle("使用卡片布局管理器");
        setBounds(100, 100, 600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        cardLayout = new CardLayout(); //创建一个卡片布局管理器对象
    }
}
```

```

cardPanel = new JPanel(cardLayout); //创建一个采用卡片布局管理器的面板对象
getContentPane().add(cardPanel, BorderLayout.CENTER); //添加到窗体中间
String[] labelNames = {"卡片 A", "卡片 B", "卡片 C"};
for (int i = 0; i < labelNames.length; i++) {
    final JLabel label = new JLabel(labelNames[i]); //创建代表卡片的标签对象
    cardPanel.add(label, labelNames[i]); //向采用卡片布局管理器的面板中添加卡片
}
final JPanel buttonPanel = new JPanel(); //创建一个按钮面板
getContentPane().add(buttonPanel, BorderLayout.SOUTH); //添加到窗体下方
String[] buttonNames = {"第一个", "前一个", "卡片 A", "卡片 B", "卡片 C", "后一个", "最后一个"};
for (int i = 0; i < buttonNames.length; i++) {
    final JButton button = new JButton(buttonNames[i]);
    button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String buttonText = button.getText();
            if (buttonText.equals("第一个"))
                cardLayout.first(cardPanel); //显示第一个卡片
            else if (buttonText.equals("前一个"))
                cardLayout.previous(cardPanel); //显示上一个卡片
            else if (buttonText.equals("卡片 A"))
                cardLayout.show(cardPanel, "卡片 A"); //显示卡片 A
            else if (buttonText.equals("卡片 B"))
                cardLayout.show(cardPanel, "卡片 B"); //显示卡片 B
            else if (buttonText.equals("卡片 C"))
                cardLayout.show(cardPanel, "卡片 C"); //显示卡片 C
            else if (buttonText.equals("后一个"))
                cardLayout.next(cardPanel); //显示下一个卡片
            else
                cardLayout.last(cardPanel); //显示最后一个卡片
        }
    });
    buttonPanel.add(button);
}
}
}

```

运行结果如图 19.11 所示。



图 19.11 使用卡片布局管理器

19.3 网格组布局管理器

 视频讲解：光盘\TM1\19\网格组布局管理器.exe

由 GridBagLayout 类实现的布局管理器称为网格组 (GridBagLayout) 布局管理器，它实现了一个

动态的矩形网格，这个矩形网格由无数个矩形单元格组成，每个组件可以占用一个或多个这样的单元格。所谓动态的矩形网格，就是可以根据实际需要随意增减矩形网格的行数和列数。

在向由 `GridBagLayout` 类管理的容器中添加组件时，需要为每个组件创建一个与之关联的 `GridBagConstraints` 类的对象，通过该类中的属性可以设置组件的布局信息，例如组件在网格组中位于第几行、第几列，以及需要占用几行几列等。

通过 `GridBagLayout` 类实现的矩形网格的绘制方向由容器决定，如果容器的方向是从左到右，则位于矩形网格左上角的单元格的列索引为 0，此时组件左上角的点为起始点；如果容器的方向是从右到左，则位于矩形网格右上角的单元格的列索引为 0，此时组件右上角的点为起始点。

下面将详细讲解 `GridBagLayout` 类中各个属性的功能和使用方法，以及在使用过程中需要注意的一些事项。

1. `gridx` 和 `gridy` 属性

这两个属性用来设置组件起始点所在单元格的索引值。需要注意的是，属性 `gridx` 设置的是 X 轴（即网格水平方向）的索引值，所以它表示的是组件起始点所在列的索引；属性 `gridy` 设置的是 Y 轴（即网格垂直方向）的索引值，所以它表示的是组件起始点所在行的索引，如图 19.12 所示。

2. `gridwidth` 和 `gridheight` 属性

这两个属性用来设置组件占用网格组的行数和列数。属性 `gridwidth` 为组件占用网格组的列数，也可以理解为以单元格为单位组件的宽度；属性 `gridheight` 为组件占用网格组的行数，也可以理解为以单元格为单位组件的高度，如图 19.13 所示。

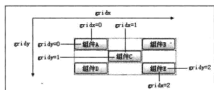


图 19.12 `gridx` 和 `gridy` 属性

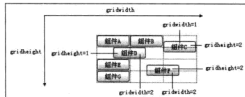


图 19.13 `gridwidth` 和 `gridheight` 属性

3. `anchor` 属性

属性 `anchor` 用来设置组件在其所在显示区域的显示位置。通常将显示区域从方向上划分为 9 个方位，分别为北方 (NORTH)、东北 (NORTHEAST)、东方 (EAST)、东南 (SOUTHEAST)、南方 (SOUTH)、西南 (SOUTHWEST)、西方 (WEST)、西北 (NORTHWEST) 和中心 (CENTER)，如图 19.14 所示。

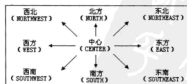
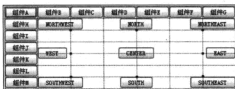


图 19.14 从方向上划分的 9 个方位

代表这 9 个方位的单词也是该类中的静态常量，可以利用这 9 个静态常量设置 `anchor` 属性，其中静态常量 `CENTER` 为默认位置。如图 19.15 所示为将组件设置为各个静态常量的效果，图中的黑点为将相应组件的 `anchor` 属性设置为 `CENTER` 时组件中心所在的位置。

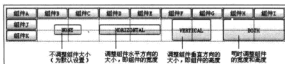
图 19.15 `anchor` 属性

说明

图 19.15 中的组件 A 至组件 M 是辅助组件，只起到占位作用。

4. `fill` 属性

属性 `fill` 用来设置组件的填充方式。当单元格显示区域的面积大于组件面积时，或者一个组件占用多个单元格时，显示组件可能不必占用所有显示区域，在这种情况下可以通过 `fill` 属性设置组件的填充方式。可以利用 4 个静态常量设置该属性，默认情况下是将该属性设置为静态常量 `NONE`，即不调整组件大小至填满显示区域；如果将该属性设置为静态常量 `HORIZONTAL`，表示只调整组件水平方向的大小（即组件宽度）至填满显示区域；如果将该属性设置为静态常量 `VERTICAL`，表示只调整组件垂直方向的大小（即组件高度）至填满显示区域；如果将该属性设置为静态常量 `BOTH`，则表示同时调整组件的宽度和高度至填满显示区域。具体效果如图 19.16 所示。

图 19.16 `fill` 属性

说明

图 19.16 中的组件 A 至组件 K 是辅助组件，只起到占位作用。

5. `insets` 属性

属性 `insets` 用来设置组件四周与单元格边缘之间的最小距离。该属性的类型为 `Insets`，`Insets` 类仅有一个构造方法 `Insets(int top, int left, int bottom, int right)`，它的 4 个入口参数依次为组件上方、左侧、下方和右侧的最小距离，单位为像素，如图 19.17 所示，默认为没有距离。

6. `ipadx` 和 `ipady` 属性

这两个属性用来修改组件的首选大小。属性 `ipadx` 用来修改组件的宽度，属性 `ipady` 用来修改组件

的高度。如果为正数，则在首选大小的基础上增加指定的宽度和高度，如图 19.18 所示。

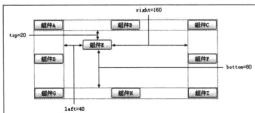


图 19.17 insets 属性

如果为负数，则在首选大小的基础上减小指定的宽度和高度，如图 19.19 所示。

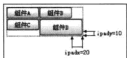


图 19.18 ipadx 和 ipady 属性为正数

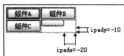


图 19.19 ipadx 和 ipady 属性为负数

7. weightx 和 weighty 属性

这两个属性用来设置网格组的每一行和每一列对额外空间的分布方式。在不设置属性 weightx 和 weighty（即采用默认设置）的情况下，当窗体调整到足够大时，将出现如图 19.20 所示的效果，组件全部聚集在窗体的中央，在组件四周出现了大片的额外空间。为了避免这种情况出现，可以通过这两个属性设置网格组的每一行和每一列对额外空间的分布方式。

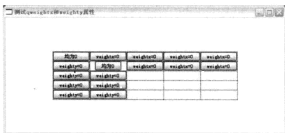


图 19.20 未设置属性 weightx 和 weighty

这两个属性的默认值均为 0，表示不分布容器的额外空间。属性 weightx 用来设置其所在列对额外空间的分布方式，如果在该列中设置了多个 weightx 属性，则取它们的最大值为该列的分布方式；属性 weighty 用来设置其所在行对额外空间的分布方式，如果在该行中设置了多个 weighty 属性，则取它们的最大值为该行的分布方式。



技巧

在设置网格组的每一行和每一列对额外空间的分布方式时，建议只设置第一行的 weightx 属性和第一列的 weighty 属性，这样会方便前期调试和后期维护。

网格组的行和列对额外空间的分布方式完全相同，下面以网格组的行为例详细讲解对额外空间的分布方式。网格组布局管理器首先计算出每一行的分布方式，即获取每一行的 `weighty` 属性的最大值；然后再计算每个最大值占所有最大值总和的百分比；最后将额外空间的相应百分比分配给对应行，如图 19.21 所示。

图 19.21 设置了属性 `weightx` 和 `weighty`



技巧 在设置网格组的每一行和每一列对额外空间的分布方式时，建议为各个属性按百分比取值。

【例 19.3】 使用网格组布局管理器管理组件，并且使用 `GridBagConstraints` 类中所有用来设置组件布局信息的属性。设计本例的出发点是对比各种设置的效果，所有注释内容均为用来对比的属性。（实例位置：光盘\TM\sl\19\3）

```
final JButton button = new JButton("A");
final GridBagConstraints gridBagConstraints = new GridBagConstraints();
gridBagConstraints.gridx = 0; //起始点为第 1 行
gridBagConstraints.gridy = 0; //起始点为第 1 列
gridBagConstraints.weightx = 10; //第 1 列的分布方式为 10%
gridBagConstraints.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(button, gridBagConstraints);
final JButton button_1 = new JButton("B");
final GridBagConstraints gridBagConstraints_1 = new GridBagConstraints();
gridBagConstraints_1.gridx = 0;
gridBagConstraints_1.gridy = 1;
gridBagConstraints_1.insets = new Insets(0, 5, 0, 0); //设置组件左侧的最小距离
gridBagConstraints_1.weightx = 20; //第 1 列的分布方式为 20%
gridBagConstraints_1.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(button_1, gridBagConstraints_1);
final JButton button_2 = new JButton("C");
final GridBagConstraints gridBagConstraints_2 = new GridBagConstraints();
gridBagConstraints_2.gridx = 0; //起始点为第 1 行
gridBagConstraints_2.gridy = 2; //起始点为第 3 列
gridBagConstraints_2.gridheight = 2; //组件占用两行
gridBagConstraints_2.insets = new Insets(0, 5, 0, 0);
gridBagConstraints_2.weightx = 30; //第 1 列的分布方式为 30%
gridBagConstraints_2.fill = GridBagConstraints.BOTH; //同时调整组件的宽度和高度
```

```

getContentPane().add(button_2, gridBagConstraints_2);
final JButton button_3 = new JButton("D");
final GridBagConstraints gridBagConstraints_3 = new GridBagConstraints();
gridBagConstraints_3.gridy = 0;
gridBagConstraints_3.gridx = 3;
gridBagConstraints_3.gridheight = 4;
gridBagConstraints_3.insets = new Insets(0, 5, 0, 5);           //设置组件左侧和右侧的最小距离
gridBagConstraints_3.weightx = 40;                             //第 1 列的分布方式为 40%
gridBagConstraints_3.fill = GridBagConstraints.BOTH;
getContentPane().add(button_3, gridBagConstraints_3);
final JButton button_4 = new JButton("E");
final GridBagConstraints gridBagConstraints_4 = new GridBagConstraints();
gridBagConstraints_4.gridy = 1;
gridBagConstraints_4.gridx = 0;
gridBagConstraints_4.gridwidth = 2;                             //组件占用两列
gridBagConstraints_4.insets = new Insets(5, 0, 0, 0);          //设置组件上方的最小距离
gridBagConstraints_4.fill = GridBagConstraints.HORIZONTAL;     //只调整组件的宽度
getContentPane().add(button_4, gridBagConstraints_4);
final JButton button_5 = new JButton("F");
final GridBagConstraints gridBagConstraints_5 = new GridBagConstraints();
gridBagConstraints_5.gridy = 2;                                 //起始点为第 3 行
gridBagConstraints_5.gridx = 0;                                //起始点为第 1 列
gridBagConstraints_5.insets = new Insets(5, 0, 0, 0);
gridBagConstraints_5.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(button_5, gridBagConstraints_5);
final JButton button_6 = new JButton("G");
final GridBagConstraints gridBagConstraints_6 = new GridBagConstraints();
gridBagConstraints_6.gridy = 2;
gridBagConstraints_6.gridx = 1;
gridBagConstraints_6.gridwidth = 2;                             //组件占用两列
gridBagConstraints_6.gridheight = 2;                            //组件占用两行
gridBagConstraints_6.insets = new Insets(5, 5, 0, 0);
gridBagConstraints_6.fill = GridBagConstraints.BOTH;           //同时调整组件的宽度和高度
getContentPane().add(button_6, gridBagConstraints_6);
final JButton button_7 = new JButton("H");
final GridBagConstraints gridBagConstraints_7 = new GridBagConstraints();
gridBagConstraints_7.gridy = 3;
gridBagConstraints_7.gridx = 0;
gridBagConstraints_7.insets = new Insets(5, 0, 0, 0);
gridBagConstraints_7.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(button_7, gridBagConstraints_7);

```

运行结果如图 19.22 所示。



图 19.22 使用网格组布局管理器

组件 C 占用两行，填充方式为全部填充；组件 D 占用 4 行，填充方式为全部填充；组件 E 占用两列，填充方式为水平填充；组件 G 占用两行两列，填充方式为全部填充。

如果将组件 G 的填充方式改为垂直填充，并在水平方向上将组件的首选宽度增加 30 像素，将显示位置设为东方，修改后组件 G 的实现代码如下：

```
final JButton button_6 = new JButton("G");
final GridBagConstraints gridBagConstraints_6 = new GridBagConstraints();
gridBagConstraints_6.gridy = 2;
gridBagConstraints_6.gridx = 1;
gridBagConstraints_6.gridwidth = 2; //组件占用两列
gridBagConstraints_6.gridheight = 2; //组件占用两行
gridBagConstraints_6.insets = new Insets(5, 5, 0, 0);
gridBagConstraints_6.fill = GridBagConstraints.VERTICAL; //只调整组件的高度
gridBagConstraints_6.ipadx = 30; //增加组件的首选宽度
gridBagConstraints_6.anchor = GridBagConstraints.EAST; //显示在东方
getContentPane().add(button_6, gridBagConstraints_6);
```

运行结果如图 19.23 所示。

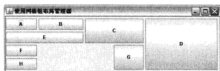


图 19.23 修改组件 G 设置后的效果

19.4 弹簧布局管理器

 视频讲解：光盘\TM\19\19.4\弹簧布局管理器.exe

由 SpringLayout 类实现的布局管理器称为弹簧（SpringLayout）布局管理器，利用该布局管理器管理组件，当改变窗体的大小时，能够在不改变组件间相对位置的前提下自动调整组件的大小，使组件依旧布满整个窗体，从而保证了窗体的整体效果。在使用该布局管理器时，需要与它的内部类 Constraints 以及 Spring 类配合使用，其中 Constraints 类用来管理组件的位置和大小，Spring 类用来创建弹簧和支架，这也是弹簧布局管理器的核心，即利用弹簧的可伸缩性动态控制组件的位置和大小。

19.4.1 使用弹簧布局管理器

弹簧布局管理器以容器和组件的边缘为操纵对象，通过为组件和容器边缘以及组件和组件边缘建立约束，实现对组件布局的管理，如图 19.24 所示。

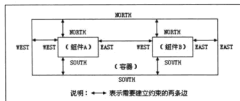


图 19.24 为容器和组件建立约束

通过方法 `putConstraint(String e1, Component c1, int pad, String e2, Component c2)` 可以为各个边之间建立约束, 该方法的入口参数说明如表 19.2 所示。

表 19.2 `putConstraint()` 方法的入口参数说明

参 数	说 明
c1	需要参考的组件对象
e1	需要参考的组件对象具体需要参考的边
c2	被参考的组件对象 (也可以是参考的组件对象所属的容器对象)
e2	被参考的组件对象具体被参考的边
pad	两条边之间的距离, 即两个组件的间距

技巧

当 c2 在 c1 的北方或西方时, pad 应为正数; 如果 c2 在 c1 的南方或东方时, pad 则应为负数; 否则两条边将重叠。

其中入口参数 e1 和 e2 可以从该类提供的静态常量中选择, 如表 19.3 所示, 这些静态常量分别表示组件相应方向的边。

表 19.3 表示组件边缘的静态常量

静 态 常 量	常 量 值	说 明
EAST	East	表示组件东侧的边
SOUTH	South	表示组件南侧的边
WEST	West	表示组件西侧的边
NORTH	North	表示组件北侧的边

下面为图 19.24 中组件 A 的北侧边和其所属容器的北侧边建立约束, 组件 A 的北侧边参考容器的北侧边, 假设两条边之间的距离为 60 像素。代码如下:

```
springLayout.putConstraint(SpringLayout.NORTH, 组件 A 对象, 60, SpringLayout.NORTH, 容器对象);
```

接着为图 19.24 中组件 B 的西侧边和组件 A 的东侧边建立约束, 假设两条边之间的距离为 60 像素。代码如下:

```
springLayout.putConstraint(SpringLayout.WEST, 组件 B 对象, 60, SpringLayout.EAST, 组件 A 对象);
```

最后为图 19.24 中组件 B 的东侧边和其所属容器的东侧边建立约束，假设两条边之间的距离为 60 像素。代码如下：

```
springLayout.putConstraint(SpringLayout.EAST, 组件 B 对象, -60, SpringLayout.EAST, 容器对象);
```

上面 3 个示例代码中，第一个演示了为组件边缘和容器边缘建立约束的方法；第二个演示了为组件边缘和组件边缘建立约束的方法；最后一个演示了当被参考的边缘在需要参考边缘的东侧时建立约束的方法，关键是需要将入口参数 pad 设为负数，否则将看不到组件 B 的东侧边缘。



说明

如果需要，也可以为图 19.24 中组件 A 的南侧边和其所属容器的北侧边建立约束；只要牢记同在水平或垂直方向上的任意两条边之间都可以建立约束即可。

【例 19.4】 使用弹簧布局管理器实现如图 19.25 所示的窗体，在调整窗体的大小后，组件仍会布满整个窗体，并且组件间的相对位置不会改变。（实例位置：光盘\TM\sl\19\4）

```
SpringLayout springLayout = new SpringLayout();           //创建弹簧布局管理器对象
Container contentPane = getContentPane();               //获得窗体容器对象
contentPane.setLayout(springLayout);                     //将窗体容器修改为采用弹簧布局管理器
JLabel topicLabel = new JLabel("主题：");
contentPane.add(topicLabel);
springLayout.putConstraint(SpringLayout.NORTH, topicLabel, 5,
    SpringLayout.NORTH, contentPane);                    //主题标签北侧——>容器北侧
springLayout.putConstraint(SpringLayout.WEST, topicLabel, 5,
    SpringLayout.WEST, contentPane);                     //主题标签西侧——>容器西侧
JTextField topicTextField = new JTextField();
contentPane.add(topicTextField);
springLayout.putConstraint(SpringLayout.NORTH, topicTextField, 5,
    SpringLayout.NORTH, contentPane);                    //主题文本框北侧——>容器北侧
springLayout.putConstraint(SpringLayout.WEST, topicTextField, 5,
    SpringLayout.EAST, topicLabel);                      //主题文本框西侧——>主题标签东侧
springLayout.putConstraint(SpringLayout.EAST, topicTextField, -5,
    SpringLayout.EAST, contentPane);                     //主题文本框东侧——>容器东侧
JLabel contentLabel = new JLabel("内容：");
contentPane.add(contentLabel);
springLayout.putConstraint(SpringLayout.NORTH, contentLabel, 5,
    SpringLayout.SOUTH, topicTextField);                 //内容标签北侧——>主题文本框南侧
springLayout.putConstraint(SpringLayout.WEST, contentLabel, 5,
    SpringLayout.WEST, contentPane);                     //内容标签西侧——>容器西侧
JScrollPane contentScrollPane = new JScrollPane();
contentScrollPane.setViewportView(new JTextArea());
contentPane.add(contentScrollPane);
springLayout.putConstraint(SpringLayout.NORTH, contentScrollPane, 5,
    SpringLayout.SOUTH, topicTextField);                 //滚动面板北侧——>主题文本框南侧
springLayout.putConstraint(SpringLayout.WEST, contentScrollPane, 5,
    SpringLayout.EAST, contentLabel);                     //滚动面板西侧——>内容标签东侧
springLayout.putConstraint(SpringLayout.EAST, contentScrollPane, -5,
    SpringLayout.EAST, contentPane);                     //滚动面板东侧——>容器东侧
```

```

JButton resetButton = new JButton("清空");
contentPane.add(resetButton);
springLayout.putConstraint(SpringLayout.SOUTH, resetButton, -5,
    SpringLayout.SOUTH, contentPane);           // “清空”按钮南侧——>容器南侧
JButton submitButton = new JButton("确定");
contentPane.add(submitButton);
springLayout.putConstraint(SpringLayout.SOUTH, submitButton, -5,
    SpringLayout.SOUTH, contentPane);           // “确定”按钮南侧——>容器南侧
springLayout.putConstraint(SpringLayout.EAST, submitButton, -5,
    SpringLayout.EAST, contentPane);           // “确定”按钮东侧——>容器东侧
springLayout.putConstraint(SpringLayout.SOUTH, contentScrollPane, -5,
    SpringLayout.NORTH, submitButton);         // 滚动面板南侧——>“确定”按钮北侧
springLayout.putConstraint(SpringLayout.EAST, resetButton, -5,
    SpringLayout.WEST, submitButton);          // “清空”按钮东侧——>“确定”按钮西侧

```

运行结果如图 19.25 所示。

调整大小后的窗体如图 19.26 所示。



图 19.25 利用弹簧布局管理器实现的窗体



图 19.26 调小后的窗体

19.4.2 使用弹簧和支柱

利用 `Spring` 类可以创建弹簧和支柱，但并不是通过构造方法实现，而是通过静态方法 `constant()` 实现。该方法有如下两个重载方法。

- ☒ `constant(int min, int pref, int max)`: 用来创建弹簧。
- ☒ `constant(int pref)`: 用来创建支柱。

在利用方法 `constant(int min, int pref, int max)` 创建弹簧时需要设置 3 个参数，分别为弹簧的最小值、首选值和最大值。最小值可以理解为了弹簧被压缩到极限时的长度；首选值可以理解为了弹簧自然放置情况下的长度；最大值可以理解为了弹簧被拉伸到极限时的长度。如果这 3 个值相等，弹簧就没有了伸缩能力，也就变成了支柱，如果是创建支柱通常利用方法 `constant(int pref)`，因为它更方便。

弹簧还有一个参数就是实际值，可以理解为了弹簧当前的长度，通过 `getValue()` 和 `setValue(int value)` 方法可以获得和设置弹簧当前的长度。

通过 `width(Component c)` 和 `height(Component c)` 方法可以快速获得一个弹簧，它们的入口参数均为组件对象，所得弹簧的最小值、首选值、最大值和实际值，分别为指定组件最小值 (`minimumSize`)、首选值 (`preferredSize`)、最大值 (`maximumSize`) 和实际值 (`size`) 的宽度或高度。

在 `Spring` 类中还提供了如下 4 个方法，均可用来根据指定条件获得一个新的弹簧。

- ☑ `max(Spring s1, Spring s2)`: 所得弹簧的最小值、首选值和最大值，均为这两个弹簧相应值中相对较大的值。
- ☑ `sum(Spring s1, Spring s2)`: 所得弹簧的最小值、首选值和最大值，均为这两个弹簧相应值的和。
- ☑ `scale(Spring s, float factor)`: 所得弹簧的最小值、首选值和最大值，均为指定弹簧相应值的 `factor` 倍。
- ☑ `minus(Spring s)`: 所得弹簧的最小值为指定弹簧最大值的负数，首选值为指定弹簧首选值的负数，最大值为指定弹簧最小值的负数。

如果需要使用弹簧建立约束，可以通过方法 `putConstraint(String e1, Component c1, int pad, String e2, Component c2)` 的重载方法 `putConstraint(String e1, Component c1, Spring s, String e2, Component c2)`，实际上前者只是简单地调用了后者。

【例 19.5】 使用弹簧和支柱动态调整按钮间的距离，其中按钮间的距离为按钮与窗体间距离的 2 倍。（实例位置：光盘\TM\sl\19\5）

```
Spring vST = Spring.constant(20);                //创建一个支柱
Spring hSP = Spring.constant(20, 100, 500);      //创建一个弹簧
JButton lButton = new JButton("按钮 L");
getContentPane().add(lButton);
springLayout.putConstraint(SpringLayout.NORTH, lButton, vST,
    SpringLayout.NORTH, getContentPane());        //“按钮 L”北侧——>容器北侧
springLayout.putConstraint(SpringLayout.WEST, lButton, hSP,
    SpringLayout.WEST, getContentPane());          //“按钮 L”西侧——>容器西侧
JButton rButton = new JButton("按钮 R");
getContentPane().add(rButton);
springLayout.putConstraint(SpringLayout.NORTH, rButton, 0,
    SpringLayout.NORTH, lButton);                 //“按钮 R”北侧——>“按钮 L”北侧
springLayout.putConstraint(SpringLayout.WEST, rButton, Spring.scale(hSP, 2),
    SpringLayout.EAST, lButton);                   //“按钮 R”西侧——>“按钮 L”东侧
springLayout.putConstraint(SpringLayout.EAST, getContentPane(), hSP,
    SpringLayout.EAST, rButton);                   //容器东侧——>“按钮 R”东侧
```

运行结果如图 19.27 所示。

可以随便调整窗体的宽度，如图 19.28 所示。

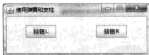


图 19.27 运行结果

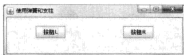


图 19.28 随便调整窗体宽度后的效果

19.4.3 利用弹簧控制组件大小

利用 `SpringLayout` 类的 `getConstraints(Component c)` 方法可以得到 `SpringLayout.Constraints` 类的对

象，通过该类的 `setWidth(Spring width)` 和 `setHeight(Spring height)` 方法可以为组件的宽度和高度添加约束。

【例 19.6】 利用弹簧控制组件大小，其中按钮的宽度由弹簧控制，高度由支柱控制。（实例位置：光盘\TM\sl\19\6）

```
Spring widthSP = Spring.constant(60, 300, 600);           //创建一个弹簧
Spring heightST = Spring.constant(60);                   //创建一个支柱
Constraints lButtonCons = springLayout.getConstraints(lButton); //获得“按钮 L”的 Constraints 对象
lButtonCons.setWidth(widthSP);                           //设置控制组件宽度的弹簧
lButtonCons.setHeight(heightST);                         //设置控制组件高度的支柱
Constraints rButtonCons = springLayout.getConstraints(rButton); //获得“按钮 R”的 Constraints 对象
rButtonCons.setWidth(widthSP);                           //设置控制组件宽度的弹簧
rButtonCons.setHeight(heightST);                         //设置控制组件高度的支柱
```

运行结果如图 19.29 所示。

可以随便调整窗体的宽度，如图 19.30 所示。

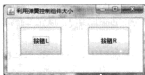


图 19.29 运行结果

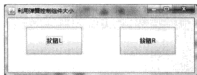


图 19.30 随便调整窗体宽度后的效果

19.5 经典范例

19.5.1 经典范例 1：制作圆形布局管理器

 视频讲解：光盘\TM\lx\19\制作圆形布局管理器.exe

尽管 Java 的 API 中实现了十余种布局管理器，但有时却不能完全满足不同用户的需求，此时可以考虑自己实现一个布局管理器。圆形是日常生活中最常见的几何形状之一，本范例演示如何实现圆形布局管理器。运行结果如图 19.31 所示。（实例位置：光盘\TM\sl\19\7）

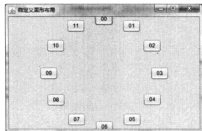


图 19.31 运行结果

编写 CircleLayout 类，该类继承了 LayoutManager。其中重点实现了 layoutContainer() 方法，该方法用来实现如何放置容器中控件的功能。代码如下：

```
public void layoutContainer(Container parent) {
    double centerX = parent.getBounds().getCenterX(); //获得容器中心的 X 坐标
    double centerY = parent.getBounds().getCenterY(); //获得容器中心的 Y 坐标
    Insets insets = parent.getInsets(); //获得容器默认边框对象
    double horizon = centerX - insets.left; //获得水平可用长度的一半
    double vertical = centerY - insets.top; //获得垂直可用长度的一半
    double radius = horizon > vertical ? vertical : horizon; //取小的为圆形半径
    int count = parent.getComponentCount(); //获得容器中控件的个数
    for (int i = 0; i < count; i++) { //依次设置所有可见控件的位置和大小
        Component component = parent.getComponent(i);
        if (component.isVisible()) {
            Dimension size = component.getPreferredSize(); //大小使用其最佳大小
            double angle = 2 * Math.PI * i / count; //获得角度的大小
            double x = centerX + radius * Math.sin(angle); //获得圆周点的 X 坐标
            double y = centerY - radius * Math.cos(angle); //获得圆周点的 Y 坐标
            //重新设置控件的位置和大小
            component.setBounds((int) x - size.width / 2, (int) y - size.height / 2, size.width, size.height);
        }
    }
}
```

19.5.2 经典范例 2：制作阶梯布局管理器

 视频讲解：光盘\TM\19\制作阶梯布局管理器.exe

尽管 Java 的 API 中实现了十余种布局管理器，有时却不能完全满足不同用户的需求。此时可以考虑自己实现一个布局管理器。阶梯是日常生活中最常见的几何形状之一。本范例演示如何实现阶梯布局管理器。运行结果如图 19.32 所示。（实例位置：光盘\TM\19\8）

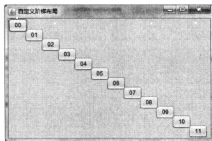


图 19.32 运行结果

编写 CircleLayout 类，该类继承了 LayoutManager。其中重点实现了 layoutContainer() 方法，该方法用来实现如何放置容器中控件的功能。代码如下：

```

public void layoutContainer(Container parent) {
    Insets insets = parent.getInsets();                //获得容器默认边框对象
    int maxWidth = parent.getWidth() - (insets.left + insets.right); //获得最大可用宽度
    int maxHeight = parent.getHeight() - (insets.top + insets.bottom); //获得最大可用高度
    int count = parent.getComponentCount();           //获得容器中控件的个数
    for (int i = 0; i < count; i++) {                 //依次设置所有可见控件的位置和大小
        Component component = parent.getComponent(i);
        if (component.isVisible()) {
            Dimension size = component.getPreferredSize(); //大小使用其最佳大小
            int x = maxWidth / count * i;                //将宽度分成 count 份根据 i 值调整 X 坐标
            int y = maxHeight / count * i;                //将高度分成 count 份根据 i 值调整 Y 坐标
            component.setBounds(x, y, size.width, size.height); //重新设置控件的位置和大小
        }
    }
}

```

19.6 本章小结

通过本章的学习，相信读者已经可以熟练地使用一些高级布局管理器，包括箱式布局管理器、卡片布局管理器、网格组布局管理器和弹簧布局管理器。至此，已经掌握了 4 种布局管理器的使用方法，通过配合使用这 4 种布局管理器，针对每个程序界面都会有多种解决方案，在设计时可以充分考虑各种解决方案的优缺点，以便从中选择一种最适合的解决方案，开发出更美观、大方、实用的程序界面。

19.7 实战练习

1. 利用本章学习的箱式、卡片和网格组布局管理器设计一个程序界面。（答案位置：光盘\TM\sl\19\9）
2. 利用弹簧布局管理器设计一个程序界面。（答案位置：光盘\TM\sl\19\10）



