SWING 入门篇

Java 学习指南系列(3)

『Java 学习指南』，居家旅行必备教程!

作者：邵发

官网 http://afanihao.cn QQ 群 495734195

目 录

第 1 章　开始 Swing

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

1.1　关于本篇

略。请参照视频。

1.2　开发环境

略。请参照视频。

1.3　第一个 GUI 程序

按视频的操作，创建一个普通的 Java 项目。添加一个类 SwingDemo，再添加一个方法，

```
private static void createGUI()
{
    JFrame frame = new JFrame("Swing Demo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    Container contentPane = frame.getContentPane();
    contentPane.setLayout(new FlowLayout());

    contentPane.add(new JLabel("Hello,World"));
    contentPane.add(new JButton("测试"));

    frame.setSize(400, 300);
    frame.setVisible(true);
```

```
    }
```

然后在 main 里运行这个方法，

```
    public static void main(String[] args)
    {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run()
            {
                createGUI();
            }
        });
    }
```

则得到一个窗口程序，如图所示。

其实这就是在 Java 快速入门与进阶 课程中的例子，不在缀述。

第 2 章　创建窗口

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

2.1　创建窗口

参照 1.3 的过程，创建一个最简单的窗口程序。

重点看一下 createGUI() 方法的实例。

```
private static void createGUI()
{
    // JFrame 指一个窗口，构造方法的参数为窗口标题
    JFrame frame = new JFrame("Swing Demo");

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container contentPane = frame.getContentPane();
    contentPane.setLayout(new FlowLayout());

    // 向内容面板里添加控件 ，如 JLabel, JButton
    contentPane.add(new JLabel("Hello,World"));
    contentPane.add(new JButton("测试"));
    // 设置窗口的其他参数，如窗口大小
    frame.setSize(400, 300);
    // 显示窗口
    frame.setVisible(true);
}
```

其中，

- JFrame，以 J 打头的类是 Swing 的基础类。JFrame 表示一个窗口。
- setDefaultCloseOperation()的表示当窗看关闭于时，同时退出进程，此方

法的作用在 Swing 高级篇里会有进一步讲解，暂时不用管。

- frame.getContentPane() 用于获取默认根容器。当创建一个 JFrame 的时候，默认是自带了一个根容器的。不过，我们一般会自己建一个 JPanel 容器作为根容器，在后面会有演示。

此时，重点放在 add()方法上。其中，

```
contentPane.add(new JLabel("Hello,World"));
```

创建一个 JLabel 控件，并添加到容器中。

相当于两步书写：

```
JLabel label = new JLabel("Hello,World");
contentPane.add ( label );
```

2.2 自定义窗口

通常，定义一个 JFrame 的子类，并在子类里完成窗口的初始化。

例如，添加一个类 MyFrame，示例如下。

```
public class MyFrame extends JFrame
{
    public MyFrame(String title)
    {
        super(title);
        // 内容面板 (ContentPane)
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

        // 向内容面板里添加控件 ，如 JLabel, JButton
        contentPane.add(new JLabel("Hello,World"));
        contentPane.add(new JButton("测试"));
    }
}
```

其中，

- super(title) ，调用父类 JFrame 的带参构造方法

在 MyFrame 的构造方法里，创建几个控件，添加到根容器中。

此时 MyFrame 就是一个自定义的窗口类。想要显示 MyFrame，只需要创建它的实例即可。示例如下。

```
private static void createGUI()
{
    MyFrame frame = new MyFrame("Swing Demo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setVisible(true);
}
```

在 Java 项目里，通常也可以写成如下形式：

```
private static void createGUI()
{
    JFrame frame = new MyFrame("Swing Demo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    frame.setVisible(true);
}
```

其中，MyFrame 是 JFrame 的子类，根据快速入门篇第 13 章**多态**的描述，这么写是完全可以的（而且是常见的）。

2.3 Swing 与 AWT

AWT（Abstract Window Toolkit），是 Java 最初提供 GUI 图形界面库。Swing，则是在 AWT 的基础上所做的进一步的优化和扩展。

也就是说，虽然我们现在讲的是 Swing 的编程，但 Swing 并不是独立于 AWT 存在的。在项目里代码，即可以看到对 java.awt.*的引用，也可以看到对 javax.swing.*的引用。

在控件名字上，有一组是 AWT 里的控件，如

Label, Button, …

而另一组是 Swing 里的控件（以 J 打头），如

JLabel, JButton，…

在本教程中，除非特殊说明，使用的都是以 J 打头的类。在书写时应该注意区分。比如，

```
contentPane.add(new Label("Hello,World"));
contentPane.add(new Button("测试"));
```

如果不小心写成这样，编译器不会报错，但运行起来会有点问题。应该改成 JLabel 和 JButton。

第 3 章　按钮事件处理

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

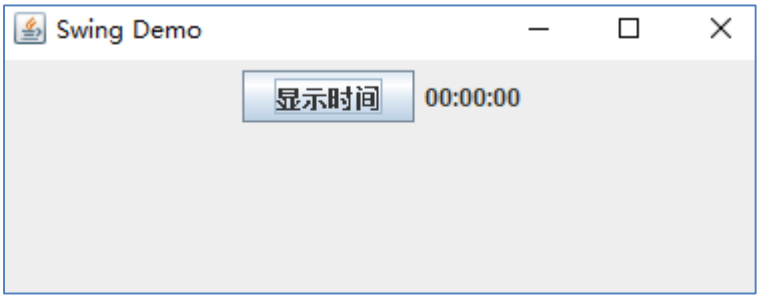
3.1　按钮点击处理

本章讨论的是按钮点击事件的处理，即点击按钮的时候，让它执行一个方法。

先构造一个简单的界面，示例如下。

```
public class MyFrame extends JFrame
{
    JLabel timeLabel = new JLabel("00:00:00");
    JButton button = new JButton("显示时间");
    public MyFrame(String title)
    {
        super(title);
        // 内容面板 (ContentPane)
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());
        // 向内容面板里添加控件
        contentPane.add(button);
        contentPane.add(timeLabel);
    }
}
```

其中，添加两个控件作为属性，分别为 timeLabel 和 button。在 main 类中创建窗口，运行，得到如下的窗口。



此时，虽然窗口里有一个按钮，但是点击这个按钮却是没有反应的。我们希望在点击按钮的时候，能执行一个方法，在方法里显示当前时间到 timeLabel 控件上。

添加一个方法，示例如下。

```
public void showTime()
{
    SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
    String timestr = sdf.format(new Date());
    timeLabel.setText( timestr );
}
```

在 showTime()方法里，取得当前时候，格式化成字符串，并设置到 timeLabel 上。如果不记得时间操作的 API，可以回顾 Java 快速入门与进阶篇的第 25 章。

目前为止，万事俱备，只欠东风。我们已经写好了 showTime()方法，缺的只是不知道怎么让系统“在点击按钮的时候调用它”。

3.2 监听器

Swing 使用监听器(Listener)的机制，来实现按钮的点击处理。一般分为三步。

(1) 定义监听器类

先创建一个类 MyButtonListener，示例代码如下。

```
private class MyButtonListener implements ActionListener
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
```

```
            System.out.println("按钮被点击...");
        }
    }
```

其中，ActionListener 是一个接口(Interface)，所以使用 implements 关键字。

(2) 创建监听器

创建一个监听器对象，交给按钮。示例如下。

```
    public MyFrame(String title)
    {
        super(title);
        …… 省略显示

        // 创建监听器对象
        MyButtonListener listener = new MyButtonListener();
        // 把监听器注册给按钮
        button.addActionListener( listener );
    }
```

(3) 运行程序

当点击按钮时，可以发现 actionPerformed()方法被运行。可以思考一下：在我们写的代码中，并没有调用 actionPerformed()方法，那么是谁调用了它呢？

实际上，当用户点按钮时，Swing 框架会自动调用这个按钮的 Listener，调用它的 actionPerformed()方法。也就是说，它是 Swing 框架自动调用的。

3.3 事件处理

下面，我们来完成真正的事件处理。也就是，当点击按钮的时候，在 JLabel 控件上显示当前的时间。

实际上，显示时间的方法 showTime()先前已经写好了。当下剩余的工作，就是看一下怎么在 actionPerformed()里调用一下外部类的 showTime()方法。

整个代码的架子是这样的，

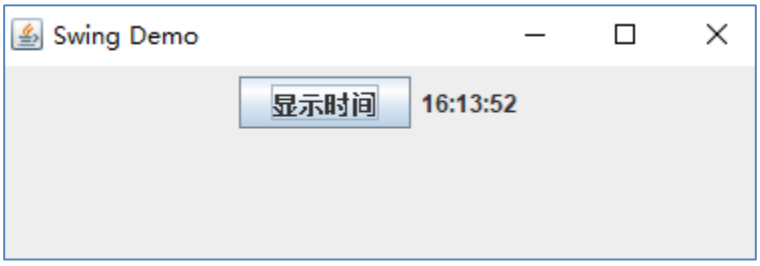
```
public class MyFrame extends JFrame
{
```

```
    public MyFrame(String title)
    {
    }
    public void showTime()
    {
        。。。显示当前时间到 JLabel。。。
    }
    private class MyButtonListener implements ActionListener
    {
        @Override
        public void actionPerformed(ActionEvent e)
        {
            MyFrame.this.showTime();
        }
    }
}
```

如何在内部类 MyButtonListener 里调用外部类 MyFrame 的方法？这个语法在 Java 入门与进阶 的第 22 章中有详细介绍。

其中，MyFrame.this 指向了当前外部对象，通过它即可以调用外面的那个 showTime()所示。

再次运行程序，在窗口里点击按钮，可以发现右侧 JLabel 控件的文本显示了当前的时间值。如图所示。



3.4 简化（匿名内部类）

当一个内部类的代码比较简短时，可以简化为匿名内部类的写法。关于匿名内部类的语法，参考 Java 快速入门与进阶的第 22 章。

代码整体的架子如下所示。

```
public class MyFrame extends JFrame
{
    public MyFrame(String title)
    {
        。。。略 。。。
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                showTime();    // 调用外部类的方法
            }
        });
    }
    public void showTime()
    {
        。。。 略 。。。
    }
}
```

在匿名类里，使用 MyFrame.this.showTime()来调用当前外部对象的方法。由于没有重名的歧义，所以可以去掉 MyFrame.this 的前缀。

3.5 简化（Lambda 表达式）

从 Java1.8 开始，引入了 Lambda 表达式的语法。使用 Lambda 表达式，可以对匿名类的写法进一步简单。

例如，使用匿名类实现的按钮点击处理，代码示例如下。

```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e)
    {
        showTime();
    }
```

```
});
```

如果使用 Lambda 表达式，则可以写得更简洁，示例如下，

```
button.addActionListener(  (e)->{
    showTime();
} );
```

其中，小括号（e）里是参数名，大括号里｛｝里是方法体。可以对照一下面的 actionPerformed()方法。

Lambda 表达式与匿名类完全等效，只是看起来简洁一些。编译器在编译时，实质还是编译成匿名内部类的。

在后续教程演示中，Swing 入门篇里一般使用匿名类的写法。随着大家对语法的掌握，在 Swing 高级篇里会逐步过渡到 Lamba 表达式的写法。

3.6 回调

回调（Callback），是一种设计上的术语。实际上，本章所介绍的 Listener 就是一种回调的设计。

回调是什么意思呢？

当我们调用 Swing 的 API 时，称为方法的调用(Call)。比如，timeLabel.setText() ，它定义了一个方法 setText()，我们调用它的方法，就称为 Call。

反过来，我们定义了一个方法 actionPerformed()，但是我们自己并未定义这个方法。Swing 框架调用了我们定义的 actionPerformed()，称为回调 (Callback)。

（注意箭头的方向）

第 4 章　简单控件

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

4.1　标签 JLabel

JLabel，称为标签控件，用于显示一个短文本。

下面演示一下 JLabel 的一些用法。

```
JLabel label = new JLabel();
contentPane.add(label);

label.setText("阿发你好官网");
label.setFont(new Font("微软雅黑", 0, 14));
label.setForeground(new Color(0, 255, 0));
label.setToolTipText("http://afanihao.cn");
```

运行程序，如图所示。

其中，

- setText()，设置显示的文本
- setFont()，设置字体，需传入一个 Font 对象
- setForeground()，设置字体颜色，需传一个 Color 对象
- setTooltipText()，设置工具提示文本

初学时，可以先照抄一下例子，然后改一改参数，尝试一下不同的效果。比如 Font 参数、Color 参数。

4.1.1 字体 Font

在 AWT/Swing 里，用 Font 类表示字体。可以设定字体名称(family)、字体大小(size)，字体风格(style)等参数。

例如，

```
Font font = new Font("宋体", Font.BOLD, 14);
```

表示设置的字体为宋体、粗体字、14 像素大小。

其中，Font.PLAIN 为普通粗细，Font.BOLD 为粗体，这是两个常量定义。

4.1.2 颜色 Color

颜色 Color 有多种指定方式。

可以直接使用一些常量，白色(Color.WHITE)、黑色(Color.BLACK)、红色(Color.RED)等。例如。

```
Color color = Color.WHITE;
```

也可以指定 RGB 值来定义一个颜色。关于 RGB 颜色值的定义，请自行百度。例如。

```
Color color = new Color(255,0,0);
```

4.1.3 无名对象

如果创建了一个对象，但是没有起名字，就可以称为无名对象。

例如，

```
panel.add(new JLabel("HelloWorld"));
```

其中，new JLabel("HelloWorld")就是创建了一个对象，然后直接把这个对象设给了 panel。这个对象是没有名字的。

也可以写成以下的样子，

```
JLabel ss = new JLabel("HelloWorld");
panel.add(ss);
```

这个 JLabel 对象有名字，名为 ss。如果对象有名字，就可以继续设置它的其他的属性参数，

ss.setFont (…)
ss.setForeground(…)

4.2 文本框 JTextField

JTextField，是一个用于显示单行文本的文本框控件。

创建一个文本框，

JTextField textField = new JTextField(20);

其中，20 表示大约的宽度参数。这个值了解一下就行，用处不大。

可以用 setText() / getText() 对其设置/取值，例如，

textField.setText("你好");

String text = textField.getText();

下面展示一个例子。在窗口里添加一个 JLabel，一个 JTextField 和一个 JButton，如下图所示。

对应的代码如下，

```
public class MyFrame extends JFrame
{
    JLabel label = new JLabel("姓名");
    JTextField textField = new JTextField(16);
    JButton button = new JButton("确定");

    public MyFrame(String title)
```

```
    {
        super(title);
        // 内容面板 (ContentPane)
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());
        // 添加控件
        contentPane.add(label);
        contentPane.add(textField);
        contentPane.add(button);
    }
}
```

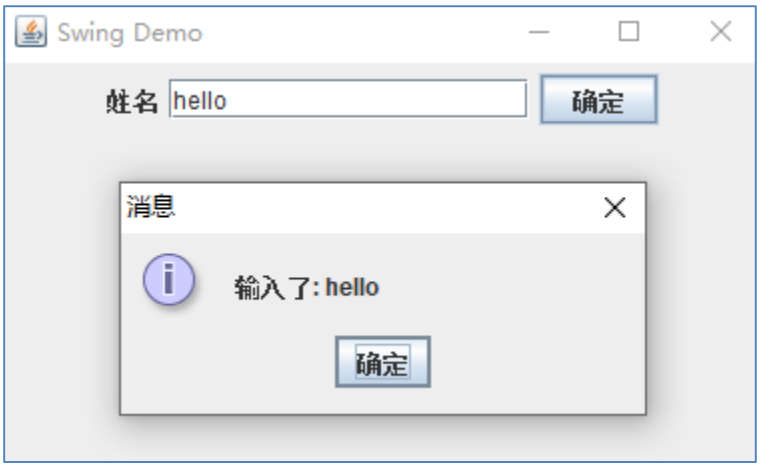
其中，把 label、textField 和 button 定义为了 MyFrame 类的属性。

下面添加按钮点击事件处理，代码架子如下所示。

```
public class MyFrame extends JFrame
{
    public MyFrame(String title)
    {
        。。。略 。。。

        button.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                onButtonOk();   // 按钮点击处理
            }
        });

    }
    private void onButtonOk()
    {
        String str = textField.getText(); // 取得文本框中的输入值
```

```
            JOptionPane.showMessageDialog(this, "输入了: " + str);
        }
    }
```

其中，JOptionPane.showMessageDialog()将弹出一个提示框。运行程序，在文本框中输入一段文字，点按钮时，提示如下。



关于 JOptionPane 的确切用法，在第 13 章中略有介绍。它不是我们的学习重点，只要了解它的功能是消息提示就足够了。

4.3 复选框 JCheckBox

JCheckBox，复选框，一个用于勾选的控件。

创建一个复选框，

```
JCheckBox cbx = new JCheckBox("我想订阅邮件通知");
```

设置其选中/取消选中，

```
cbx.setSelected(true);
```

设置其显示的文本，

```
cbx.setText("我想订阅");
```

下面在例子中展示一下。构造一个界面，在里面添加一个 JCheckBox 和一个 JTextField。如图所示。

相关代码如下，

```
public class MyFrame extends JFrame
{
    JCheckBox cbx = new JCheckBox("我想订阅邮件通知");
    JTextField email = new JTextField(16);

    public MyFrame(String title)
    {
        super(title);
        // 内容面板 (ContentPane)
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());
        // 添加控件
        contentPane.add(cbx);
        contentPane.add(email);

        cbx.setSelected(true); // 默认选中
        email.setToolTipText("输入邮箱地址");
    }
}
```

4.3.1 复选框事件处理

和普通按钮类似，使用 JCheckBox. addActionListener()可以添加一个监听器，当复选框状态改变时监听器被调用。示例代码如下，

```
cbx.addActionListener( new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // 如果未选中，则禁用文本框
        if(cbx.isSelected())
            email.setEnabled(true);
        else
            email.setEnabled(false);
    }
});
```

其中，JTextField.setEnabled() 可以禁用/使能文本框，当文本框被禁用时，文本框不允许输入。

以上代码的业务逻辑是，当复选框被勾选时，将 email 文本框允许输入。而当复选框没有勾选时，就不需要留一个 email 了。

4.4 下拉列表 JComboBox

JComoboBox，表示一个下拉列表控件。下拉列表是一个选择控件，点箭头时弹出一个菜单供用户选择。本例实现一个界面，如图所示。

怎么创建一个下拉列表控件呢？

```
JComboBox<String> colorList = new JComboBox<>();
```

可以看到，JComboBox 是一个泛型类，尖括号里指定是每一项(Item)的数据类型。在这里，用 String 作为 Item 的数据类型。

使用 addItem (T value) 来添加一项，其中 T 就是泛型指定类型参数。向下拉列表中添加 3 项数据，示例如下。

```
colorList.addItem("红色");
colorList.addItem("蓝色");
colorList.addItem("绿色");
```

这里的 addItem()只能传递一个 String 参数，因为在定义的时候已经指定了 JComboBox<String>，类型参数为<String>。

4.4.1 JComoboBox API

使用索引，可以访问下拉列表中的每一项。

```
T value = colorList.getItemAt( index) ;  // 获取第几项
int count = colorList.getItemCount();  // 一共多少项
```

索引从 0 开始计算，0 表示第一项，1 表示第二项。。。除非特殊说明，编程领域的索引都是从 0 开始的。

此处的 T，指的是泛型参数<String>，在真正写的时候应该替换成相应的泛型参数类型，如 String value = colorList.getItemAt(0) 。

JComoboBox 有两组 API，一组是按索引 Index 操作，另一组是按值 Value 操作。按索引操作的 API 如下所列，

- getSelectedIndex() ，获取当前选中项的索引
- setSelectedIndex(index) ，设置当前选中项
- remove(index)，删除一项

按值操作的 API 所列，

- T value = getSelectedItem()，获取当前项的值
- setSelectedItem(value) ，设置当前选中值，内部会按值用 equals()比较
- remove(value)，删除一项，内部会按值用 equals()比较

4.4.2 JComoboBox 事件处理

使用 addActionListener()可以为其添加监听器，当下拉列表选择事件发生后，进入事件处理。示例代码如下，

```
colorList.addActionListener( new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e)
    {
        updateTextColor();
    }
});
```

当下拉列表被选择后，调用 updateTextColor()进行处理，其代码如下，

```
private void updateTextColor()
{
    // 获取选中项的值
    String item = (String)colorList.getSelectedItem();
    // 根据选中的颜色，设置 JLabel 文字颜色
    Color color = null;
    if( item.equals( "红色"))
        color = Color.RED;
    else if(item.equals("蓝色"))
        color = Color.BLUE;
    else if(item.equals("绿色"))
        color = Color.GREEN;
    sampleText.setForeground(color);
}
```

其中，根据选中项的值，作出不同的处理。其实也可以用 getSelectedIndex()，获取选中的索引、再根据索引进行判断。

4.4.3 JComoboBox 自定义 Item

JComboBox<T>是一个泛型，参数 T 不一定是 String，也可以是任意其他类型。比如，可以添加一个类 ListOption 作为数据项的类型。示例代码如下。

```
private static class ListOption     // static class，静态内部类
{
    public String text;
```

```
        public Color color;
        public ListOption(String text, Color color)
        {
            this.text = text;
            this.color = color;
        }
        @Override
        public String toString()    // 重定 toString()用于列表项的显示
        {
            return "[" + this.text + "]" ;
        }
    }
```

其中注意两点，

- ListOption 被定义为 static class，即静态内部类。因为它是一个比较独立的类，仅仅表示一个数据项。静态内部类的语法，参考 Java 入门与进阶的 22 章。
- ListOption 要重写 toString()方法，以便一个 ListOption 对象可以转成 String，作为列表项的显示。

下面，定义一个下拉列表控件，

```
JComboBox<ListOption> colorList = new JComboBox<>();
```

其中，<ListOption>表示列表项的类型为 ListOption。

往里面添加几项数据，创建 ListOption 对象并传给 addItem()，

```
colorList.addItem( new ListOption("红色", Color.RED));
colorList.addItem( new ListOption("绿色", Color.GREEN));
colorList.addItem( new ListOption("蓝色", Color.BLUE));
```

在下拉列表显示时，并自动调用列表项的 toString()方法，作为列表项的显示，效果如下：

其中，[红色] 字样，正是 ListOption.toString()的输出格式。

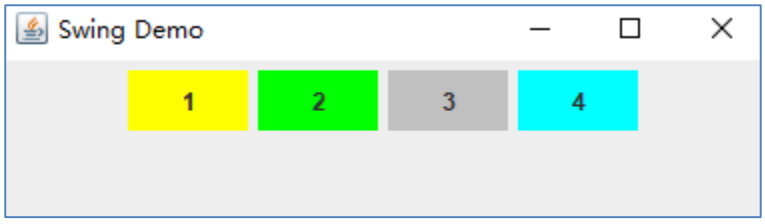
在下拉列表的事件处理时，

```
private void updateTextColor()
{
    ListOption item = (ListOption)colorList.getSelectedItem();
    sampleText.setForeground(item.color);
}
```

此时，colorList.getSelectedItem()的返回值，是一个 ListOption 类型的对象。这就是泛型的意思，当定义时指定<ListOption>参数，那么后面的 addItem(T)等方法中的 T 都成为 ListOption 类型。

4.5 （练习）彩色标签

本节将实现一个彩色标签的例子，效果如下所示。



在此窗口里，并排添加了 4 个彩色背景的标签。

用 JLabel 可以实现这种彩色的标签，例如，

```
JLabel a1 = new JLabel("1");
a1.setOpaque(true); // 设置背景为不透明 （默认透明 )
a1.setBackground(Color.YELLOW); // 设置背景色
a1.setPreferredSize(new Dimension(60,30)); // 设置最佳尺寸
a1.setHorizontalAlignment(SwingConstants.CENTER); // 设置水平对齐
```

其中，

- Dimension 用于指定尺寸，宽度和高度单位为像素材
- setOpaque(true)，表示设为不透明的

提示：如果要 setBackground()设置背景色，必须要调用 setOpaque(true)，颜色才能生效。

接下来，如何向窗口里添加 4 个彩色标签呢？似乎有点麻烦，

```
JLabel a1 = new JLabel("1");
a1.setOpaque(true);
a1.setBackground(Color.YELLOW);
a1.setPreferredSize(new Dimension(60,30));
a1.setHorizontalAlignment(SwingConstants.CENTER);

JLabel a2 = new JLabel("2");
a2.setOpaque(true);
a2.setBackground(Color.GREEN);
a2.setPreferredSize(new Dimension(60,30));
a2.setHorizontalAlignment(SwingConstants.CENTER);

JLabel a3 = new JLabel("3");
a3.setOpaque(true);
a3.setBackground(Color.LIGHT_GRAY);
a3.setPreferredSize(new Dimension(60,30));
a3.setHorizontalAlignment(SwingConstants.CENTER);

JLabel a4 = new JLabel("4");
a4.setOpaque(true);
a4.setBackground(Color.CYAN);
a4.setPreferredSize(new Dimension(60,30));
a4.setHorizontalAlignment(SwingConstants.CENTER);
```

也就是说，需要依次创建 4 个 JLabel 对象，并按照完全相同方式对其设置。这样是可以实现的，但是并不科学，如此重复冗余的代码是可以优化的。

优化的办法，是创建一个自定义标签类，继承于 JLabel 即可。示例代码如下。

```
private static class ColorfulLabel extends JLabel
{
    public ColorfulLabel(String text, Color bgColor)
    {
        super(text);
```

```
            setOpaque(true);
            setBackground(bgColor);
            setPreferredSize(new Dimension(60,30));
            setHorizontalAlignment(SwingConstants.CENTER);
        }
    }
```

其中，ColorfulLabel 构造方法带了 2 个参数，text 表示要显示的文本，bgColor 表示背景色。

然后就可以用简洁的代码来创建 4 个彩色标签，

```
JLabel a1 = new ColorfulLabel("1", Color.YELLOW);
JLabel a2 = new ColorfulLabel("2", Color.GREEN);
JLabel a3 = new ColorfulLabel("3", Color.LIGHT_GRAY);
JLabel a4 = new ColorfulLabel("4", Color.CYAN);
```

再把这 4 个标签添加到窗口里，便实现了前面的彩色标签的实例效果。

第 5 章　布局管理器

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

5.1　流布局 FlowLayout

当一个容器里添加了多个子控件时，每个子控件的大小和位置，是由布局管理器(Layout Manager)决定的。先前的例子中，都是设置了一个 FlowLayout 类型的管理器。示例如下：

```
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
```

也可以写成下面这种样子，

```
Container contentPane = getContentPane();
LayoutManager layout = new FlowLayout(FlowLayout.LEFT);
contentPane.setLayout(layout);
```

其中，显式地创建了一个 FlowLayout 对象，并设置给 contentPane。所以，在显示的时候 contentPane 会把子控件布局的事性交给这个布局管理器。

注意，在语法上 LayoutManager 是一个接口，FlowLayout 类实现了 LayoutManager 这个接口。

(提示：在实际项目中一般不会使用 FlowLayout，了解即可)

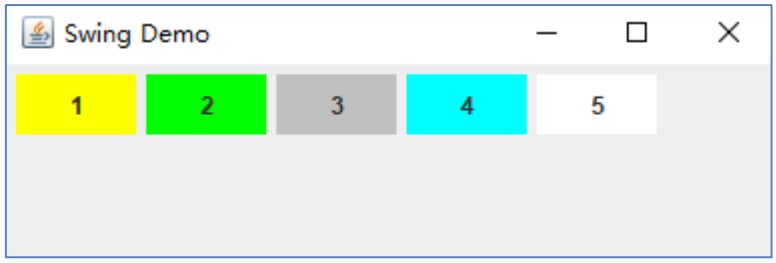
5.1.1 FlowLayout 特点

流式布局很简单，就是从左到右、从上到下，依次对每一个子控件进行布局。比如，在容器里添加 5 个控件，示例如下。

```
JLabel a1 = new ColorfulLabel("1", Color.YELLOW);
JLabel a2 = new ColorfulLabel("2", Color.GREEN);
```

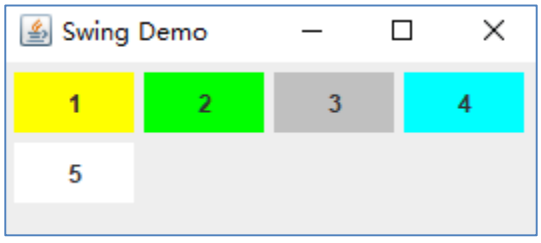
```
JLabel a3 = new ColorfulLabel("3", Color.LIGHT_GRAY);
JLabel a4 = new ColorfulLabel("4", Color.CYAN);
JLabel a5 = new ColorfulLabel("5", Color.WHITE);

contentPane.add(a1);
contentPane.add(a2);
contentPane.add(a3);
contentPane.add(a4);
contentPane.add(a5);
```

运行程序，在窗口里显示 5 个标签控件。如图所示。



可以看到，它是从左到右依次显示的。

当窗口不够宽时，一行里面显示不了所有控件的时候，它会设法换行显示。示例效果如下。

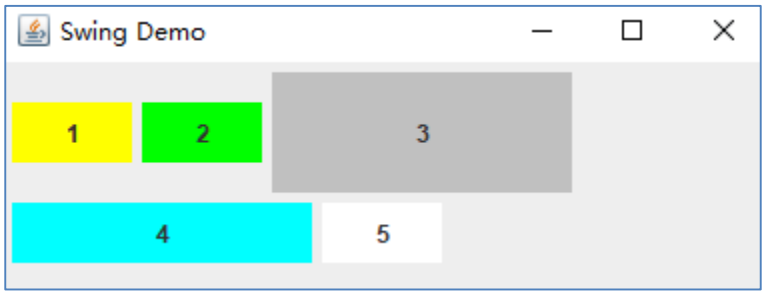


5.1.2 FlowLayout 设置控件尺寸

那么，每一个控件的大小是怎么决定的呢？通过 setPreferredSize(w,h) 可以设置子控件的显示大小，示例如下。

```
a3.setPreferredSize(new Dimension(150,60));
a4.setPreferredSize(new Dimension(150,30));
```

将 a3 设置为 150x60 像素，a4 设置为 150x30 像素。运行程序，效果如下。



可以看到，标签 3 和标签 4 的尺寸显示发生了明显变化。

也就是说，在 FlowLayout 布局中，使用 setPreferredSize()可以设置每一个子控件的显示大小。

5.2 边界布局 BorderLayout

BorderLayout，边界布局，是一个实际项目中常用的布局管理器。使用边界布局，将整个容器划分为 5 个区域：上、下、左、右、中。

例如，给容器设置一个 BorderLayout 布局管理器，

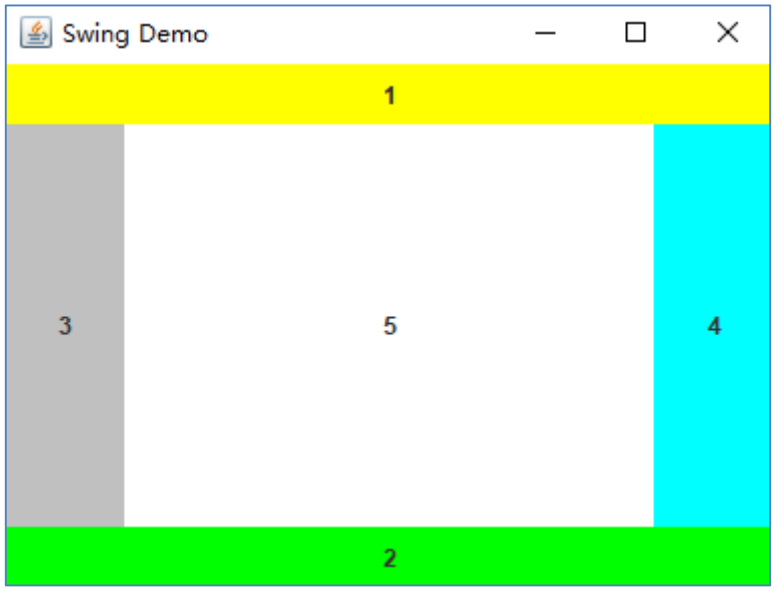
```
contentPane.setLayout(new BorderLayout());
```

然后，添加 5 个标签控件，

```
JLabel a1 = new ColorfulLabel("1", Color.YELLOW);
JLabel a2 = new ColorfulLabel("2", Color.GREEN);
JLabel a3 = new ColorfulLabel("3", Color.LIGHT_GRAY);
JLabel a4 = new ColorfulLabel("4", Color.CYAN);
JLabel a5 = new ColorfulLabel("5", Color.WHITE);

contentPane.add(a1, BorderLayout.PAGE_START);   // 上
contentPane.add(a2, BorderLayout.PAGE_END);     // 下
contentPane.add(a3, BorderLayout.LINE_START);   // 左
contentPane.add(a4, BorderLayout.LINE_END);     // 右
contentPane.add(a5, BorderLayout.CENTER);       // 中
```

其中，在 add()添加子控件时，要指定一个布局参数。PAGE_START 为上，PAGE_END 为下，LINE_START 为左，LINE_END 为右，CENTER 为中。

运行程序，显示效果如下图所示。



(提示：BorderLayout 在实际项目中较为常用)

在边界布局时，各个控件的尺寸如何设定呢？

对于上下两边，其宽度恒定为 100%，高度可以 setPreferredSize()调整。例如，

```
a1.setPreferredSize(new Dimension(0,80));
```

其中，将 a1 的高度设为 80 像素。由于它处于上边界，所以宽度参数传任何值都是没有意义的，它的宽度总是占据 100%的宽度。此处传入宽度参数为 0。

对于左右两边，其高度恒定为 100%，宽度则可以调整。例如，

```
a3.setPreferredSize(new Dimension(100,0));
```

同样的道理，a3 位于左边界，其高度参数没有意义。

对于中间的控件，总是占据剩余的中部区域空间，无法调整大小。

5.3 盒布局（deprecated）

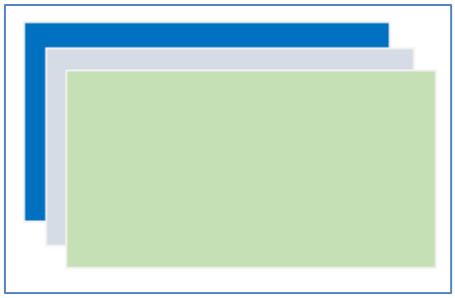
本节已废弃。请跳过。

5.4 盒（deprecated）

本节已废弃。请跳过。

5.5 卡片布局 CardLayout

CardLayout，卡片布局，就是像卡片一样叠在一起的布局。示意图所示。



给容器设置为卡片布局，

```
CardLayout cardLayout = new CardLayout();
pane.setLayout ( cardLayout );
```

添加几个控件（卡片），

```
pane.add ( c1,   "name1");
pane.add ( c2,   "name2");
pane.add ( c3,   "name3");
```

这样，就向容器里添加了 3 张卡片，每个卡片都关联一个名字。在显示的时候，需要指定卡片的名称。例如，想显示控件 c3，则调用

```
cardLayout.show(pane, "name3");
```

其中，在调用 show()时传入目标卡片的名字 name3 即可。

(提示：CardLayout 在 Swing 入门/高级篇中均不太常用，不作为重点)

提示：以后要用到的几种常见的布局器。

- BorderLayout，边界布局器
- AfRowLayout 横向布局
- AfColomunLayout 纵向布局
- AfAnyWhere 自由位置布局

其中，Af 打头的几个布局器在第 7 章中介绍。

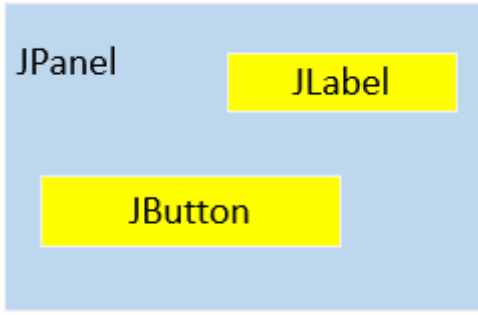
第 6 章　自定义布局

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

6.1　窗口坐标

当在一个容器（Container）里添加多个子控件（Component）时，容器要负责子控件的布局。所谓布局，就是决定把每个控件放在什么位置、宽高是多大。

常见的容器类为 JPanel，在 JPanel 里可以添加多个子控件。比如，可以在一个 JPanel 面板里添加一个 JLabel 控件、一个 JButton 控件。如图所示。

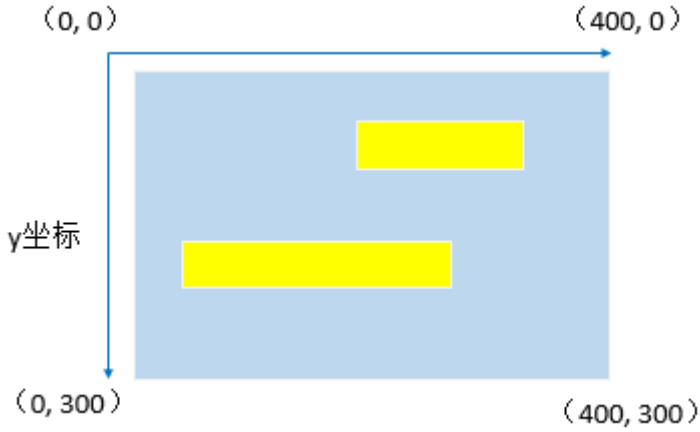


那么，当窗口显示时，容器要决定这两个控件显示在什么位置（Position）、以及宽度和高度(Size)。在 Swing 里，使用 Rectangle 类来描述一个子控件的位置和大小。例如，

```
Rectangle rect = new Rectangle(30,30,200,100);
```

则此矩形描述的是左上角坐标为(30,30)，宽度 200 像素，高度 100 像素。那么，这个坐标是规定的呢？

一般地，在计算机 GUI 编程时，都是规定左上角为最小值（0，0），右下角为最大值。比如，一个 400x300 像素的窗口，它的左上角坐标为（0，0）。水平向右则 X 坐标增加，竖直向下则 Y 坐标增加。如下图所示。



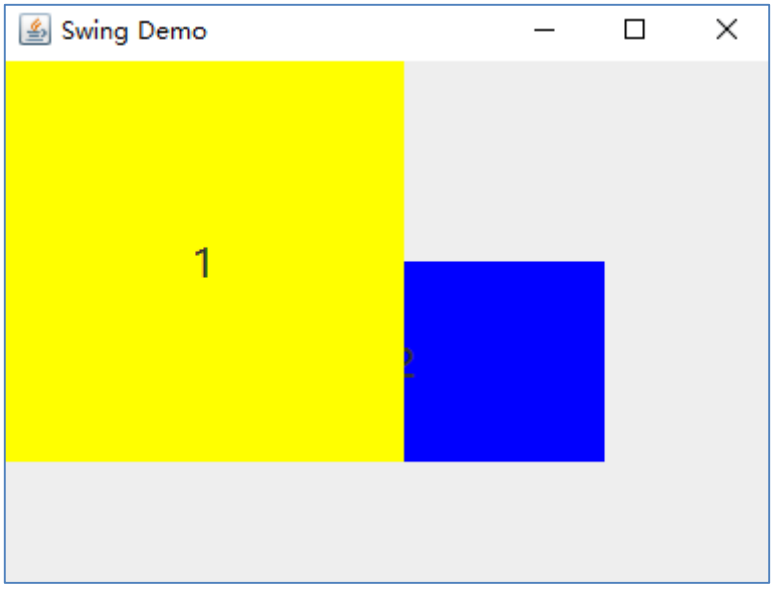
下面用一个小例子，来增强对坐标的理解。新建一个窗口，添加 2 个控件。使用 setBounds()方法来设定每个控件的显示位置。

```
public MyFrame(String title)
{
    super(title);

    Container root = this.getContentPane();
    root.setLayout(null); // 取消其布局器

    // 增加 2 个控件
    JLabel a1 = new ColorfulLabel("1", Color.yellow);
    JLabel a2 = new ColorfulLabel("2", Color.blue);
    root.add(a1);
    root.add(a2);

    a1.setBounds(new Rectangle(0,0, 200,200));
    a2.setBounds(new Rectangle(100,100, 200,100));
}
```

其中，a1.setBounds(new Rectangle(0,0, 200,200)) 用来设置控件 a1 的显示位置，用一个 Rectangle 指定它的位置。也可以写为两行，示例如下：

```
Rectangle rect = new Rectangle(0,0,200,200);
a1.setBounds(rect);
```

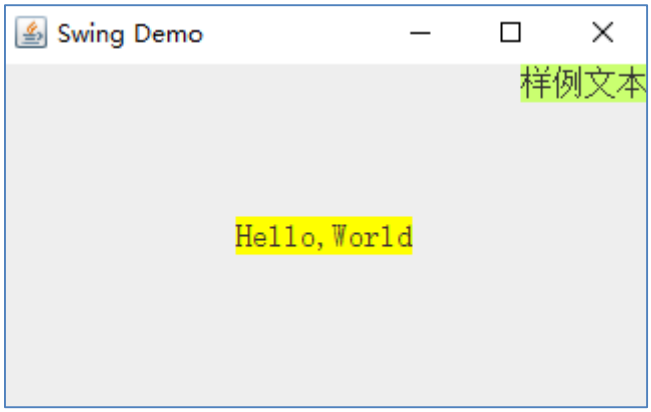
运行程序，效果如下图所示。



在此例中，1 号标签显示于(0,0, 200,200)这个矩形位置。2 号标签显示于(100,100,200,100)这个矩形位置。由于这两矩形有位置重叠，所以显示的时候有一定的覆盖关系（注：到底谁覆盖谁，不是本节的重点）。

6.2 创建布局器

布局器 (Layout Manager)，负责对容器内的子控件进行布局。比如上一章所介绍了 FlowLayout 或者 BorderLayout，都是布局器类。有的界面需要实现自定义的布局，比如，用自定义布局器实现如下图所示的效果。



在此容器中，有两个标签控件，一个控件总是显示在右上角，另一个控件总是显示在中央。当窗口大小改变时，它们的相对位置会自动重新测算。这种效果就需要使用自定义布局的技术。

下面介绍怎么样来自定义一个布局器类。

自定义布局器时，需要创建一个类，实现 LayoutManager 接口。例如，在 MyFrame 里添加一个内部类 SimpleLayout，示例如下，

```
private class SimpleLayout implements LayoutManager
{
    public void addLayoutComponent(String name, Component comp){}
    public void removeLayoutComponent(Component comp){}
    public Dimension preferredLayoutSize(Container parent){}
    public Dimension preferredLayoutSize(Container parent){}
    public void layoutContainer(Container parent){ 这是核心方法 }
}
```

在 LayoutManager 里要求重写 5 个方法，其中的 layoutContainer()是核心方法，在这个方法里对容器中的每一个控件进行动态的测算和布局。

定义这个 SimpleLayout 布局器类之后，就可以和普通的布局器一样来使用它了。例如，

```
JPanel root =new JPanel();
LayoutManager layoutMgr = new SimpleLayout();
root.setLayout(layoutMgr);
```

或者直接写为，

```
root.setLayout ( new SimpleLayout () );
```

于是 root 拥有了一个 SimpleLayout 对象作为布局器，当窗口大小改变（或其他情况时），内部会自动调用 SimpleLayout.layoutContainer() 来重新布局。

6.2.1 布局测算

在 layoutContainer()方法里，要负责对容器里的子控件进行布局测算，决定每一个控件显示的位置、宽度和高度。

在本例中，SimpleLayout 是 MyFrame 的内部类，所以在 layoutContainer()里可以直接访问 MyFrame 里定义的属性。比如，在 MyFrame 里定义了 a1、a2 两个标签控件，下面看一下怎么对它们进行布局。

示例代码如下所示。

```
        public void layoutContainer(Container parent)
        {
            int w = parent.getWidth(); //  父窗口的宽度  width
            int h = parent.getHeight(); //  父窗口的高度  height
            System.out.println("父容器: " + w + ", " + h);

            // a1  显示在中间，以  Preferred Size  作为大小
            if(a1.isVisible())
            {
                //  得取该控件所需的显示尺寸
                Dimension size = a1.getPreferredSize();
                //System.out.println(size);
                int x = (w - size.width) / 2;
                int y = (h - size.height) /2 ;

                //  在设置子控件位置时，其坐标是相对于父窗口的
                // (0,0)  就是父窗口的左上角
                a1.setBounds(x, y, size.width, size.height);
            }

            // a2  显示在右上角
            if(a2.isVisible())
            {
                Dimension size = a2.getPreferredSize();
                //System.out.println(size);
                int x = (w - size.width);
                int y = 0;
                a2.setBounds(x, y, size.width, size.height);
            }
        }
    }
```

其中，首先获得了容器本身的宽度和高度，然后获取控件的显示尺寸。经过

计算，则 a1 控件显示在容器中央，a2 控件显示在容器右上角。

6.2.2 布局的运行

布局器是怎么运行的呢？首先，需要给容器设置一个布局器。例如，

```
root.setLayout( layoutMgr );
```

随后，当容器大小改变时，内部会自动调用布局器重新布局。具体来说，布局器的 layoutContainer()方法会被内部调用。

这个调用过程由 Swing 框架内部触发，我们是看不到的。为了验证这一点，可以在 layoutContainer()里添加一些 pringln()打印输出。当调整窗口大小时，可以观察到这些打印输出。

6.3 手动布局

像上一节那样，在布局器里手工指写每一个控件的大小和位置，称为手动布局。在手动布局时，一般需要创建一个内部类，重写 LayoutManager 接口的 5 个方法，在 layoutContainer()里测算每个控件的大小和位置。

在 LayoutManager 接口里，规定了 5 个方法，但是在手动布局时，要关注的只有 layoutContainer()这一个方法，其他几个方法只需要默认置空就可以了。

为了简化代码，可以建立一个 AfSimpleLayout 类，

```
public abstract class AfSimpleLayout implements LayoutManager
{
    public void addLayoutComponent(String name, Component comp){}
    public void removeLayoutComponent(Component comp){}
    public Dimension preferredLayoutSize(Container parent){}
    public Dimension minimumLayoutSize(Container parent){}
}
```

其中，AfSimpleLayout 重写了 4 个方法，还缺少一个方法。

当需要手动布局时，只需要继承 AfSimpleLayout，然后一下 layoutContainer()方法即可。例如，

```
    private class MyCustomLayout extends AfSimpleLayout
    {
```

```
        @Override
        public void layoutContainer(Container parent)
        {
            int w = parent.getWidth(); // 父窗口的宽度 width
            int h = parent.getHeight(); // 父窗口的高度 height
            。。。 略 。。。
        }
    }
```

其中，MyCustomLayout 继承于 AfSimpleLayout，只需要重写 layoutContainer()这一个方法，其他 4 个方法在父类中已经有了定义。

需要注意的是，语法上 AfSimpleLayout 需要声明为 abstract。为什么呢？因为它继承于 LayoutManager 接口，还缺一个抽象方法 layoutContainer()没有定义。

6.4 线性布局

线性布局，指的是在水平方向上依次布局，或者上竖直方向上依次布局。这是在实际项目中最常见的布局方式。这里提供两个方便易用的布局器，分别用于水平和竖直方式上的布局。

6.4.1 水平布局

各个子控件在水平方向上依次排列，每个控件的宽度可以单独指定。在下面的示意图中，容器里添加 4 个控件，水平排列。

要实现这样的水平布局，可以使用 AfXLayout 或 AfRowLayout ，两者等效。例如，

```
root.setLayout(new AfXLayout(8));     // 子控件之间的间距为 8 像素
root.add(a1, "80px");
root.add(a2, "30%");
root.add(a3, "auto");
```

```
root.add(a4, "1w");
```

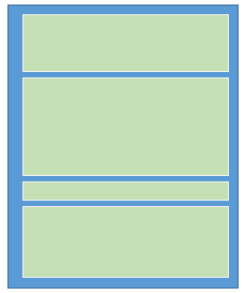
其中，宽度参数有 4 种形式，例如，

- "80px" 固定占 80 像素
- "20%" 固定占总空间的 20%
- "auto" 采用控件的 Preferred Size
- "1w" 按权重分配剩余空间

例如，一个控件为"2w"，一个控件为"3w"，则它们将按 2:3 比例瓜分剩余的空间。

6.4.2 竖直布局

将各个子控件从上到下依次布局，每个控件的高度可以单独指定。看下面的示意图，是把 4 个控件添加到容器里，并在竖直方向上依次排列。



要实现这样的竖直布局，可以使用 AfYLayout 或 AfColumnLayout，两者等效。例如，

```
root.setLayout(new AfYLayout(8));
root.add(a1, "80px");
root.add(a2, "30%");
root.add(a3, "auto");
root.add(a4, "1w");
```

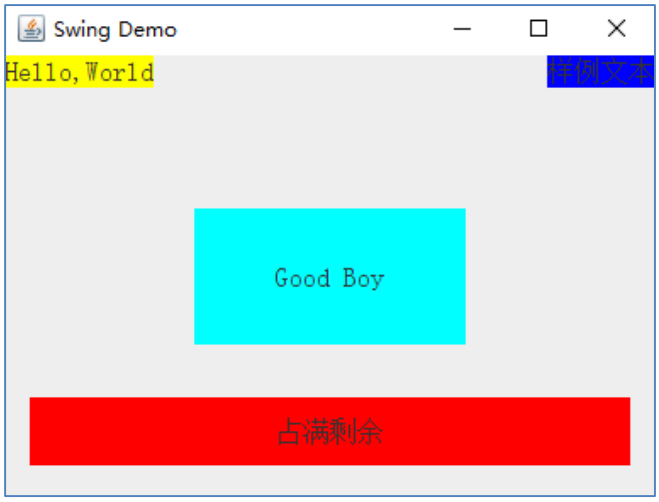
其中，和水平布局器一样，也是采用 像素值、百分比、自动和权值四种参数。

提示：AfXLayout 和 AfYLayout 的内部实现较为复杂，不要求掌握。只要会使用它们即可。

提示：在本教程中，AfXLayout 和 AfRowLayout 都表示水平布局。由历史问题，这种布局器一直没有太满意的命名，所以这两种命名在后续的演示中都有可能出现。

6.5 自由位置布局

下面再介绍另一个布局器，AfAnyWhere，自由位置布局器。使用这个布局器，可以将一个控件布置在指定的位置。先看一下演示效果，如图所示。



在此图中，一个控件在左上角，一个在右上角。一个在中央位置，一个在中下的位置。使用 AfAnyWhere 布局器，可以很轻松地实现这种自由地、不规则的布局方式。

6.5.1 AfAnyWhere

如何使用呢？首先，给容器设置一个 AfAnyWhere 作为布局器，例如。

```
panel.setLayout(new AfAnyWhere());
```

然后，在向容器里添加控件的时候，要同时指定一个 AfMargin 作为布局参数。例如，

```
panel.add(label, new AfMargin(-1, 15, 20, 15));
```

其中，AfMargin 用于规定该控件的上、左、下、右四个边距。如果边距为-1，表示是的自动计算。

在该例中，(-1,15,20,15)分别表示上左下右的间距的值。上为-1，左为 15，下为 20，右为 15。由于左右间距都为 15，所以水平方向上呈现拉伸的效果。由于上间距为-1，下间距为 20，所以竖直方向上靠下显示。

为了便于调用，定义了 10 个常用的方位作为常量。分别是：

```
public static final AfMargin FULL = new AfMargin(0, 0, 0,0);
public static final AfMargin TOP_LEFT = new AfMargin(0, 0, -1, -2);
public static final AfMargin TOP_CENTER = new AfMargin(0, -1, -1, -1);
public static final AfMargin TOP_RIGHT = new AfMargin(0, -1, -1, 0);
public static final AfMargin CENTER_LEFT = new AfMargin(-1, 0, -1, -1);
public static final AfMargin CENTER = new AfMargin(-1, -1, -1, -1);
public static final AfMargin CENTER_RIGHT = new AfMargin(-1, -1, -1, 0);
public static final AfMargin BOTTOM_LEFT = new AfMargin(-1, 0, 0, -1);
public static final AfMargin BOTTOM_CENTER = new AfMargin(-1, -1, 0, -1);
public static final AfMargin BOTTOM_RIGHT = new AfMargin(-1, -1, 0, 0);
```

比如，要将一个控件显示在右上角，

```
panel.add(label, AfMargin.TOP_RIGHT);
```

或者写成，

```
panel.add(label, new AfMargin(0, -1, -1, 0));
```

6.5.2 自由布局示例

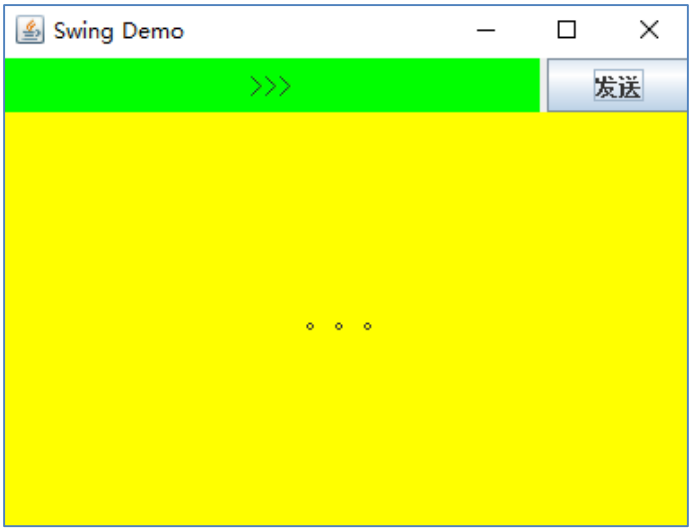
所以，使用 AfAnyWhere 就可以很容易实现前面图中的演示效果。示例代码如下。

```
root.setLayout(new AfAnyWhere());
root.add(a1, AfMargin.TOP_LEFT);
root.add(a2, AfMargin.TOP_RIGHT);

root.add(a3, AfMargin.CENTER); //  中间
a3.setPreferredSize(new Dimension(160,80));

root.add(a4, new AfMargin(-1, 15, 20, 15));
a4.setPreferredSize(new Dimension(0,40));
```

同样的，AfAnyWhere 作为一个工具类出现，大家会使用它就行了。它的内部实现有些难度，目前不要求研究和掌握。

6.6 （练习）综合布局

以后项目中，最常用的布局器就是 BorderLayout、AfXLayout(AfRowLayout)和 AfYLayout (AfColumnLayout)。下面用这几个布局器，实现这样的界面布局。如图所示。



在此例中，有 3 个控件，它们不是简单地水平或竖直显示。

设计思路：整体上可以设为 BorderLayout，中间显示了一个大的控件。上部可视为一个子容器，在子容器里添加 2 个控件、并设置水平布局。

示例代码如下所示。

```
root.setLayout(new BorderLayout());   // 总体上使用 BorderLayout
root.add(a1, BorderLayout.CENTER);   // 中间加一个控件
if( true )
{
    JPanel p = new JPanel();     // 上面是一个组合
    p.setLayout(new AfXLayout(4));
    p.add(a2,"1w"); // 权重为 1，水平方向占满剩余空间
    p.add(sendButton, "80px"); // 水平方向固定 80 像素宽度

    root.add(p,   BorderLayout.PAGE_START);
}
```

其中，JPanel 是一个在布局里通常的容器，可以视为一个子面板，整个大的界面就是 JPanel 一层一层地组合而成的。

第 7 章　边框

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

7.1　使用边框

下面介绍边框(Border)的实现方法。在 Swing API 里，与边框相关的定义大部分都在 javax.swing.border.*这个包下面。

Interfaces

Border

Classes

AbstractBorder
BevelBorder
CompoundBorder
EmptyBorder
EtchedBorder
LineBorder
MatteBorder
SoftBevelBorder
StrokeBorder
TitledBorder

其中，Border 为一个接口类，下面所列的是实现类，如 LineBorder 表示一种线条状的边框。

比如，给控件 a2 设置一个线条边框，示例如下。

```
Border border2 = new LineBorder(Color.BLUE, 4);
a2.setBorder(border2);
```

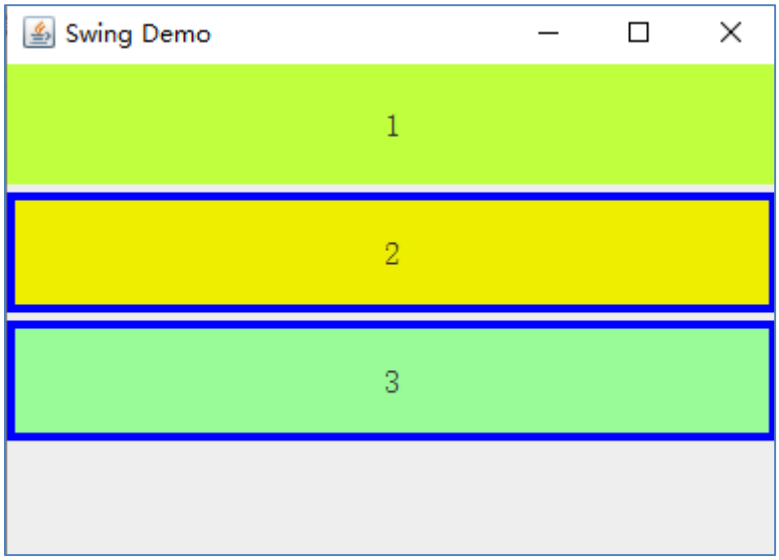
其中，在创建 LineBorder 对象时，指定颜色和宽度两个参数。然后通过 setBorder()方法设置给控件。

Swing 也提供了一个快捷工具类 BorderFactory，里面提供了一个工具方法(public static 方法)，便于快速地创建各种类型的边框。

上面的例子也可以写为，

```
Border border2 = BorderFactory.createLineBorder(Color.BLUE, 4);
a2.setBorder(border2);
```

在 Java API 中，以 Factory 命名的类可以理解为”工厂”。所谓的 BorderFactory，就可以理解为用于生产 Border 的工厂。运行代码，效果如图所示。



图中，2 号标签和 3 号标签均设置了一个蓝色的边框。

7.2 各种样式的边框

Swing API 里提供了各种类型的边框的实现，种类很多，其实在实际项目中用不到这么多种边框，所以没有必要一一尝试。

这些边框大体上可以分为四类：

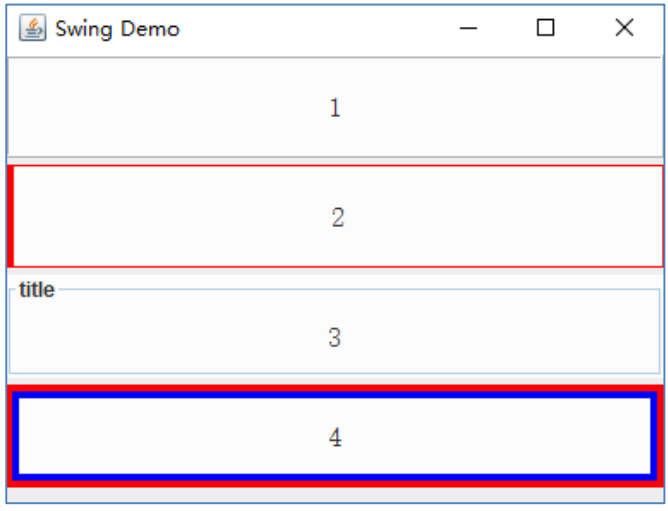
- 简单线条边框 Simple
- 特种边框 Matte
- 带标题边框 Titled
- 复合边框 Compound

在创建边框，使用 BorderFactory 提供的工具方法就可以了。下面给出一个简单的展示。

```
// 简单线条边框
JLabel a1 = new ColorfulLabel("1", new Color(0xfcfcfc));
root.add(a1, "60px");
a1.setBorder( BorderFactory.createEtchedBorder(EtchedBorder.RAISED) );
// 特种边框 (4 条边分别可设置)
JLabel a2 = new ColorfulLabel("2", new Color(0xfcfcfc));
root.add(a2, "60px");
a2.setBorder( BorderFactory.createMatteBorder(1, 5, 1, 1, Color.red) );
// 带标题的边框
JLabel a3 = new ColorfulLabel("3", new Color(0xfcfcfc));
root.add(a3, "60px");
a3.setBorder( BorderFactory.createTitledBorder("title") );
// 复合边框
JLabel a4 = new ColorfulLabel("4", new Color(0xfcfcfc));
root.add(a4, "60px");
Border outer = BorderFactory.createLineBorder(Color.RED, 4);
Border inner = BorderFactory.createLineBorder(Color.BLUE, 4);
Border compound = BorderFactory.createCompoundBorder(outer, inner);
a4.setBorder(compound);
```

注意，创建边框的方法了解即可，不需要记忆。当在项目中需要用到时，回过到头来参考一个官方示例就可以了。

运行上述代码，效果如下图所示。



在此例中，分别演示了 4 个边框。1 是简单的线条边框(Line Border)。2 是上左下右宽度不同的边框(Matte Border)。3 是带标题的边框（Titled Border）。4 是双层复合边框(Compound Border)。

7.2.1 复合边框

下面着重看一下复合边框的使用。所谓复合边框，就是把两个 Border 合二为一。示例代码如下。

```
Border outer = BorderFactory.createLineBorder(Color.RED, 4);
Border inner = BorderFactory.createLineBorder(Color.BLUE, 4);
Border compound = BorderFactory.createCompoundBorder(outer, inner);
a4.setBorder(compound);
```

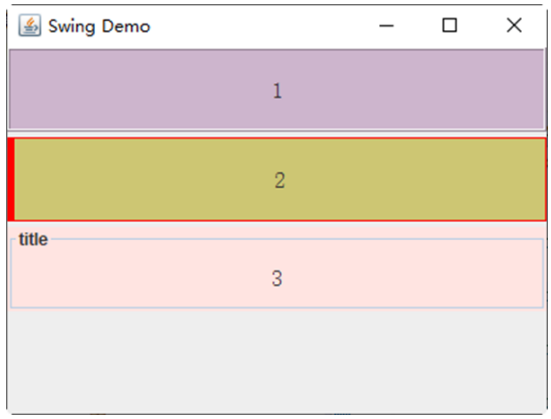
其中，使用 createCompoundBorder()即可以将边框 outer、inner 合起来。把外层边框称为 Outer Border，内部边框称为 Inner Border。

运行示例代码，得到的是一个外层为红色、内层为蓝色的双层复合边框。

同理，使用 createCompoundBorder()多次合成，就可以得到一个三层边框。在下一节中，将进一步演示复合边框的应用。

7.3 边距与填充

下面讨论一下边距的设置。在布局时，一般为了让界面更清爽，需要给容器或控件设置边距。比如，下面的布局在没有设置边距的情况下，显得比较拥挤。

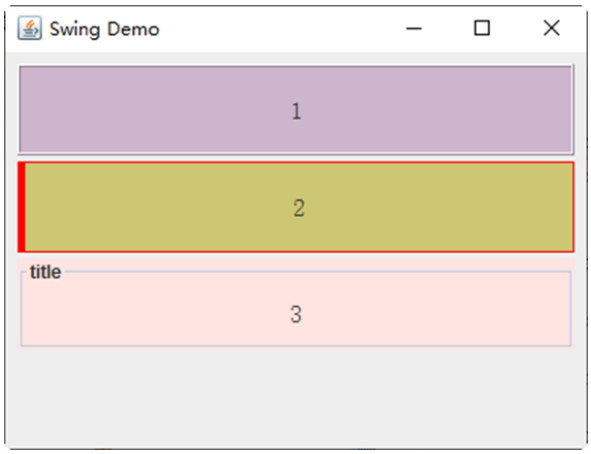


在此窗口里，添加了 3 个控件，内容没有什么问题，唯一的问题就是布局太靠边了。如果能够容器设置一个内边距，则看起来会清爽得多。

在 Swing 里，边距也是用 Border 的 API 来实现的。例如，

```
Border padding = BorderFactory.createEmptyBorder(8, 8, 8, 8);
root.setBorder(padding);
```

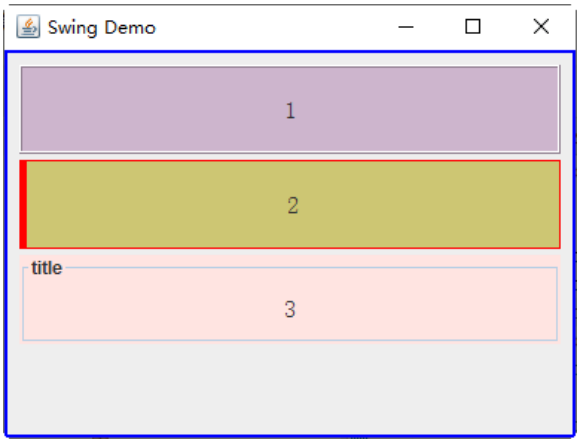
这样就给容器 root 设置一个 Empty 边框，在显示上就是边距的效果，上左下右四个边的边距都是 8 个像素。效果如下所示。



利用复合边框的技术，可以实现把边框和边距一并设置。示例如下。

```
Border padding = BorderFactory.createEmptyBorder(8, 8, 8, 8);
Border border = BorderFactory.createLineBorder(Color.BLUE, 2);
border = BorderFactory.createCompoundBorder(border, padding);
root.setBorder( border );
```

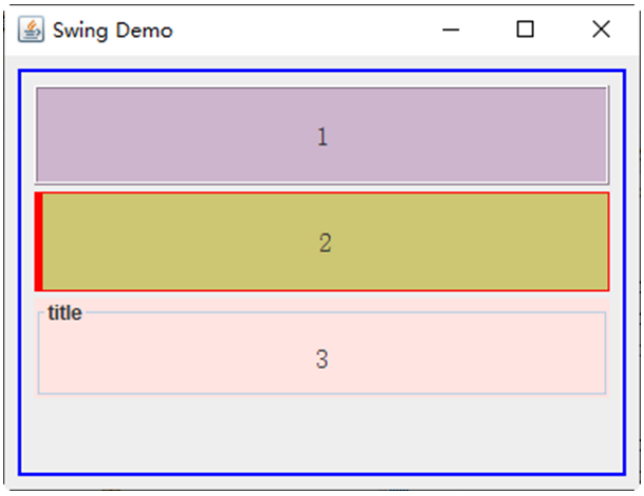
其中，创建了两个 Border。一个称为 Padding，表示内边框。一个为线条边框。把它们合起来，就实现了既有边线又有内边距的效果。效果如图所示。



进一步地，使用多层复合的技术，可以同时设置外边距(Margin)、边框、内边距(Padding)，示例代码如下。

```
Border padding = BorderFactory.createEmptyBorder(8, 8, 8, 8);
Border border = BorderFactory.createLineBorder(Color.BLUE, 2);
border = BorderFactory.createCompoundBorder(border, padding);
Border margin = BorderFactory.createEmptyBorder(8, 8, 8, 8);
border = BorderFactory.createCompoundBorder(margin, border);
root.setBorder( border );
```

运行代码，效果如下所示。



其中，边线之外的间距称为 Margin，表示该控件与上层容器的间距。边线之内的间距称为 Padding，表示该控件与其内容的间距。

7.4 AfBorder

对上一节的内容作一下封装，得到工具类 AfBorder，用于快速地设置边框和边距。

```
public class AfBorder
{
    public static void addPadding(JComponent c, int size) {…}
    public static void addPadding(JComponent c, int top, int left, int bottom, int right){…}
    public static void addMargin(JComponent c, int size){…}
    public static void addMargin(JComponent c, int top, int left, int bottom, int
```

right){…}

public static void addOuterBorder(JComponent c, Border outerBorder){…}

public static void addInnerBorder(JComponent c, Border innerBorder){…}

}

其中，

- JComponent 是 Swing 里一般控件的父类，比如，JTextField, JLabel, JButton 都直接或间接地继承自 JComponent。
- addAdding()，用于设置内边距
- addMargin()，用于设置外边距
- addOuterBorder()，用于设置外边框
- addInnerBorder()，用于设置内边框

使用 AfBorder 这个类，就可以快速的设置边距。比如，

```
AfBorder.addPadding(root, 8);  // 上、左、下、右均为 8 像素
```

或者

```
AfBorder.addPadding(root, 16, 8, 0, 8);  // 上、左、下、右不同边距
```

由于 addPadding()是静态方法（public static），所以在调用时直接用类名来调用即可。这种像工具一样直接使用的方法，可以称为工具方法。

在 AfBorder 类里重载了两个版本的 addPadding()方法，一个用于设置统一的边距值，上左下右的边距值相同。另一个用于设置上、左、下、右不同的边距。

```
// 设置内间距
public static void addPadding(JComponent c, int size)
{
    addPadding(c, size, size, size, size);
}

// 设置内间距
public static void addPadding(JComponent c, int top, int left, int bottom, int right)
{
    Border border = BorderFactory.createEmptyBorder(top, left, bottom, right);
    addInnerBorder(c, border);
}
```

```
    // 附加一个内边框
    public static void addInnerBorder(JComponent c, Border innerBorder)
    {
        Border border = c.getBorder();
        if(border != null)
        {
            // 如果原来有一个边框，则进行复合
            border = BorderFactory.createCompoundBorder(border, innerBorder);
            c.setBorder( border );
        }
        else
        {
            c.setBorder(innerBorder);
        }
    }
```

可以看到，这三个方法之间存在着调用关系。第 1 个 addPadding()里调用了第 2 个 addPadding()，后者又调用了 addInnerBorder()。

7.5 AfPanel

下面再提供一个工具类，AfPanel，用于实现快速的界面布局。AfPanel 继承于 JPanel，提供了一些快捷方法。其代码结构如下。

```
public class AfPanel extends JPanel
{
    public AfPanel padding( int size){…}
    public AfPanel padding( int top, int left, int bottom, int right){…}
    public AfPanel margin( int size){…}
    public AfPanel margin( int top, int left, int bottom, int right){…}
    public void addOuterBorder(JComponent c, Border outerBorder){…}
    public void addInnerBorder(JComponent c, Border innerBorder){…}
    public AfPanel preferredSize(int w, int h){…}
    public AfPanel preferredWidth(int w){…}
```

```
    public AfPanel preferredHeight(int h){…}
}
```

其中，

- AfPanel 继承于 JPanel，所以 JPanel 原有的方法还是可以照常使用的；
- padding() 用于快速的设置内间距
- margin() 用于快速的设置外边距
- preferredWidth() 用于快速地设置最佳宽度
- preferredHeight() 用于快速地设置最佳高度

在界面布局时，设置间距和宽高是很常用的操作，所以在 AfPanel 里作了一定程序的封装，使得调用的过程更简洁一下。

比如，

```
AfPanel p = new AfPanel();
p.setLayout( new AfXLayout( 4));   // setLayout()照常使用
p.padding(10); //  内间距
p.preferredHeight(80); //  最佳高度
```

7.5.1 链式调用的形式

AfPanel 里的几个方法都被设计为允许链式调用。Java 里的链式调用，指是的几个方法连续调用的情况。形如，

```
AfPanel p = new AfPanel();
p.padding(10).preferredHeight(80);
```

为什么可以这么写呢？观察一下 padding()方法的实现，

```
public AfPanel padding( int top, int left, int bottom, int right)
{
    AfBorder.addPadding(this, top,left, bottom,right);
    return this;     //  为什么返回 this？为什么不是 void 返回？
}
```

这个方法为什么返回当前对象 this 呢？这就是链式调用语法的关键。下面详细说明一下链式调用的语法。

7.5.1 链式调用的语法

(1) 普通形式调用

```
public class Point
{
    public int x;
    public int y;
    public void setX(int x)
    {
        this.x = x;
    }
    public void setY(int y)
    {
        this.y = y;
    }
}
```

调用形式:

```
Point p = new Point();
p.setX ( 120);
p.setY ( 180);
```

(2) 链式调用

修改 Point 类的 setX()和 setY()方法，使之支持链式调用。

```
public class Point
{
    public int x;
    public int y;
    public Point setX(int x)
    {
        this.x = x;
        return this;   // 要支持链式调用，必须返回当前对象
    }
```

```
    public Point setY(int y)
    {
        this.y = y;
        return this;   // 要支持链式调用，必须返回当前对象
    }
}
```

此时，Point 的 setX()和 setY()是支持链式调用的，形如，

```
Point p = new Point();
p.setX(120).setY(180);
```

为什么可以连着写呢？

因为 p.setX(120) 是有返回值的，返回的是当前这个 Point 对象。

相当于:

(?) .setY(180) ，其中 ？ 为 p.setX()的返回值

进一步地，还可以写成这样，

```
Point p = new Point().setX(120).setY(180);
```

相当于：

```
( new Point() ) .setX(120).setY(180);     // ( ? ).setX().setY()
( (new Point() ).setX(120) ) . setY(180);      // ( ? ).setY()
```

总结一下，要支持链式调用，则这个方法的返回值必须是当前对象 this。

第 8 章　图标

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

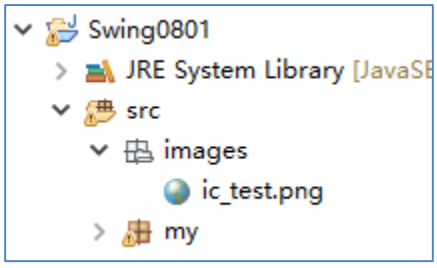
8.1　使用图标

在 Swing 控件里，JLabel 和 JButton 都可以设置显示一个图标。与图标相关的有两个类型，Icon 和 ImageIcon。其中，Icon 其实是一个接口，而 ImageIcon 是一个具体实现的类。

下面来介绍 Swing 程序中的图标的使用。

要显示图标，首先得准备好一张图标文件。可以从 iconfont.cn 上去自己下载一张图标，供学习测试使用。默认支持的静态图片格式为 jpg/jpeg 或 png 格式。

比如，下载了一个 ic_test.png，放在项目里什么位置呢？可以在 src 目录下新建一个 images 包，然后把图片文件拷贝到 images 包下面。如图所示。



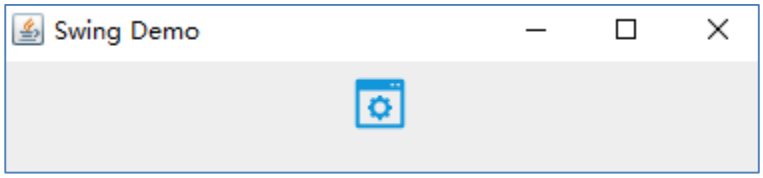
也就是说，在 package 下面不但可以有 Java 的源文件，也可以放图片文件，和任意格式的文件。

下面看一下怎么在代码里加载这个图片文件，示例代码如下。

```
URL url = getClass().getResource("/images/ic_test.png");
Icon icon = new ImageIcon(url);
iconLabel.setIcon(icon);
```

其中，通过 getClass().getResource(…)来得到一个图片的路径 URL，注意图片路径的写法：/images/ic_test.png。

ImageIcon 类代表一个图标，创建一个 ImageIcon 对象，设置给 JLabel 控件，即可以实现图标的显示。运行程序，效果如下所示。



8.2 资源文件

一般来说，Java 项目的 src 目录下放的是源代码文件(*.java)，在 bin 目录下放的是程序文件(*.class)。现在出现了一个特殊情况，src/images/目录下放了一个 ic_test.png 的图片文件。

对于 src 下的图片文件，Eclipse 会自动拷贝一个副本到 bin 的相同目录下。可以检查一下项目目录：

src/images/ic_test.png

bin/images/ic_test.png

我们把 bin 目录下的图片文件，称为资源文件。根据一个资源文件的包路径，可以转成一个 URL，例如，

URL url = getClass(). getResource("/images/icon/open.png")

或者

URL url = MyFrame.class.getResource("/images/icon/open.png")

在程序运行的时候，Swing 会从 bin 目录下按路径查找目录资源文件。确切地讲，是从环境变量 Classpath 路径中查找的，而 bin 目录是在 Classpath 里的。关于 Classpath，在高级语法篇中略有介绍，这里大概理解一下就行。

总之，Swing 可以把 bin 目录下资源文件加载进来，知道这一点就够了。

8.3 本地文件

在 Java 里访问一个文件时，通常有三种位置。

(1) 资源文件，放在 src/ 下

(2) 本地文件，放在本地某个目录下，但不是在 src 目录下

(3) 网络文件，通过 HTTP 访问的一个网络资源，在后面的网站系列教程里会介绍。

在 Swing 里使用图片时，最常用的方式就是资源文件或本地文件。

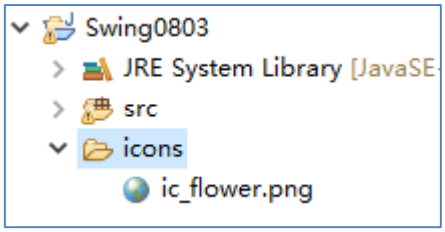
比如说，一个文件放在 C:/icons/test.png，

```
String filePath = "C:/icons/test.png";
Icon icon = new ImageIcon(filePath);
```

传入一个文件路径，就能加载得到 ImageIcon 图标。文件路径可以是绝对路径，如"C:/icons/test.png"。也可以是相对路径，如 icons/test.png。关于绝对路径和相对路径的概念，参考 Java 入门与进阶的第 26 章。

下面演示一下相对路径的用法。

首先，在项目下创建一个目录 icons 目录，然后把图片文件拷贝到此目录中。如图所示。



注意，icons 目录和 src 是同一级目录。

```
JLabel label2 = new JLabel();
Icon icon = new ImageIcon("icons/test.png");
label2.setIcon( icon );
```

在 Eclipse 里运行项目时，默认的工作目录就是项目所在的目录。

可以简单地区分一下，放在 src 下的就是资源文件，放在普通目录下的就是本地文件。

8.4 （练习）工具按钮

略。请对照视频练习。

第 9 章　自定义控件

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

9.1　自定义控件

各种各样的自定义技术，是 Swing 篇的重点。因为 Swing API 本身并没有太大的意思，只有通过各种自定义的训练，才能有效提升程序设计的能力。

前面已经学过了 Swing 自带的一些控件，例如 JButton，JLabel 等。那么，是不是可以自己定义一些控件呢？当然是可以的。一般的 GUI 技术，都是允许自定义控件的，Swing 也不例外。使用自定义技术，不但可以实现 JButton 这种控件，还可以比它做得更好、更炫酷。

在 Swing 里怎么自定义一个控件呢？一般是继承于 JPanel 类。例如，

```
public class MyPanel extends JPanel
{
    @Override
    protected void paintComponent(Graphics g)
    {
    }
}
```

其中，MyPanel 继承于 JPanel，所以它就是一个自定义控件。它重写了 paintComponent()，在这个方法里面可以做一些控件绘制的工作。

提示：不要把 paintComponent() 误写为 printComponent()

不同的控件都有不同的样子，比如 JButton 和 JCheckBox，是完全不同的形状。那现在自定义一个控件 MyPanel，它应该长什么样子，是由我们自由定义的。比如，可以把它绘制为红色，示例代码如下。

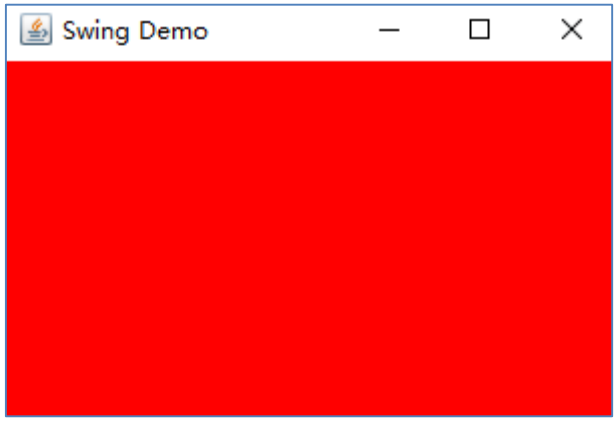
```
protected void paintComponent(Graphics g)
{
    int width = this.getWidth();    // 本控件的宽度
    int height = this.getHeight();  // 本控件的高度
    g.clearRect(0, 0, width, height); // 清除显示
    g.setColor(new Color(0xFF0000)); // 设置填充色为红色
    g.fillRect(0, 0, width, height); // 绘制填充一个矩形区域
}
```

然后创建一个 MyPanel 控件，添加到主界面布局中，代码如下。

```
Container root = this.getContentPane();
root.setLayout( new BorderLayout());
MyPanel p = new MyPanel();
root.add(p, BorderLayout.CENTER); // 添加到容器
```

可以看到，创建一个 MyPanel 控件，和普通控件一样，它也可以被加到容器中，被布局器管理。

运行代码，效果如下所示。



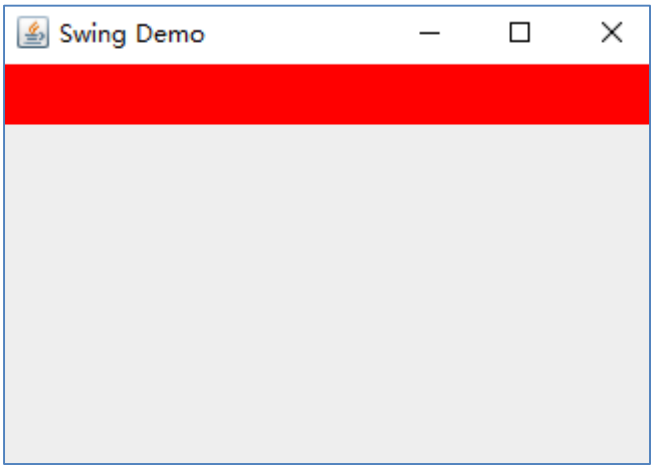
此时，自定义控件被添加到 BorderLayout 的中间部位，控件显示为红色。之所以显示为红色，是因为当窗口显示时，Swing 框架内部会自动调用控件的 paintComponent()方法进行绘制。

需要注意的是，paintComponent()是框架内部自动调用的，我们在代码里没有显式地调用它。此方法可能被调用多次，什么时候调用，调用几次，都是由框架自己决定的。比如，当窗口改变大小的时候，控件会被重新绘制，paintComponent()方法会被调用。

由于 MyPanel 继承于 JPanel，所以 JPanel 原有的方法都是可以照常使用的。例如，设置它的最佳尺寸，示例如下。

```
MyPanel p = new MyPanel();
root.add(p, BorderLayout.PAGE_START);   // 显示在上面
p.setPreferredSize(new Dimension(0,30));   // 设置最佳高度为 30px
```

再次运行代码，效果如下所示。



可以发现，在主界面上方(PAGE_START)显示了一个红色的控件，高度为 30 像素。它就是我们创建的自定义控件。

9.2 RGB 颜色

RGB，(Red 红，Green 绿，Blue 蓝)，是计算机编程里一种常见的颜色表示法。准确地讲，RGB 是一种颜色空间(Color Space)，除此之外还有别的颜色表示法。

RGB 颜色使用红绿蓝三原色分量来表示一个颜色，例如，

黑色 (0, 0, 0)

白色 (255, 255, 255)

红色 (255, 0, 0)

其中，每一个分量最小为 0，最大为 255。比如，黑色(0,0,0)表示全暗，而白色(255,255,255)表示红绿蓝全为最亮，合成之后即为白色。通过不同的 RGB 组合可以表示不同的彩色。

在 Swing 里，使用 Color 类表示颜色，常见的创建方式有：

```
Color color = Color.red;
```

Color color = new Color (255, 0, 0);

Color color = new Color(0xFF0000);

其中，Color.RED 是一个常量。

在定义颜色值时，可以传入 R、G、B 分别指定，如 new Color(255,0,0)。也可以直接传一个十六进制的整数值，如 new Color(0xFF0000)。

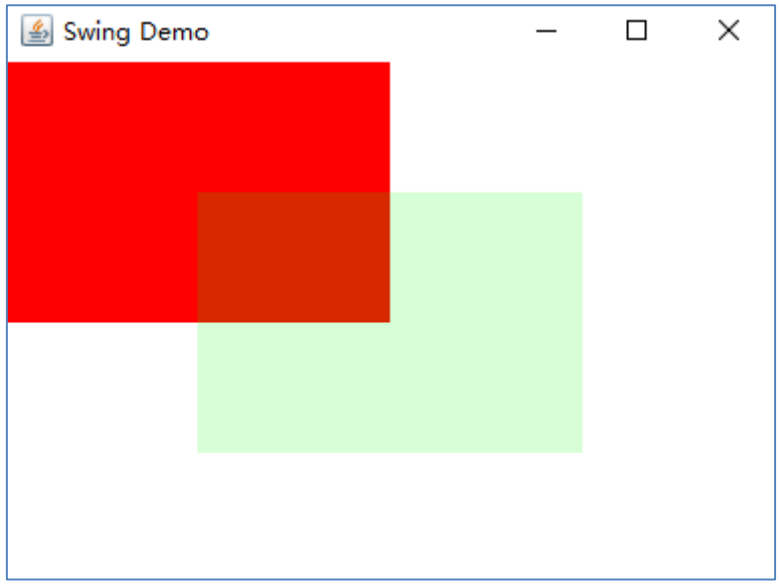
9.2.1 透明度

使用 RGBA 定义，可以定义一个半透明的颜色，其中 A 表示 Alpha。透明度最小为 0，表示全透明。最大为 255，表示完全不透明。例如，

```
Color c1 = new Color(0,255,0, 40);   // 最后一个值 40，表示透明度
Color c2 = new Color(0x2800FF00, true); // 最高位 0x28 为透明度
```

其中，第 2 种写法理解起来有点难度，不要求掌握。

下面看一下透明绘制的效果。



在这个控件里绘制了 2 个矩形，第 2 个淡淡的矩形就是使用了半透明的颜色。示例代码如下。

```
protected void paintComponent(Graphics g)
{
    int width = this.getWidth();
    int height = this.getHeight();
```

```
        g.setColor(new Color(255,0,0));
        g.fillRect(0, 0, width/2, height/2);    // 第一个矩形
        g.setColor( new Color(0,255,0, 40));    // Alpha 值为 40
        g.fillRect(width/4, height/4, width/2, height/2);    // 第二个矩形
    }
```

一般来说，后绘制的图形会叠加在先前的图形之上，形成遮盖效果。但是使用半透明色绘制时，就是半透的效果，不会完全遮盖。

9.3 绘制几何图形

在自定义的控件里，可以绘制很多种东西。比如，可以绘制文本，几何形状，或者是图片。下面来了解一下几何形状的绘制。

在 Swing 里，支持以下几何形状的绘制：

- Line 直线
- Rect 矩形 (包含正方形)
- Oval 椭圆 (包含圆)
- Polygon 多边形 (含三角形)
- Arc 圆弧 / 扇形

关于控件的绘制，内容非常多且复杂，所以在 Swing 入门篇里仅做大致了解，更深入的内容都在 Swing 高级篇讲解。

在绘制时，需使用 Graphics 类提供一些绘制方法。查阅 Java 的官方文档，可以发现 Grpahics 类下面的绘制方法大致可以分为两类，一类是 draw 打头，一类以 fill 的打头。例如，

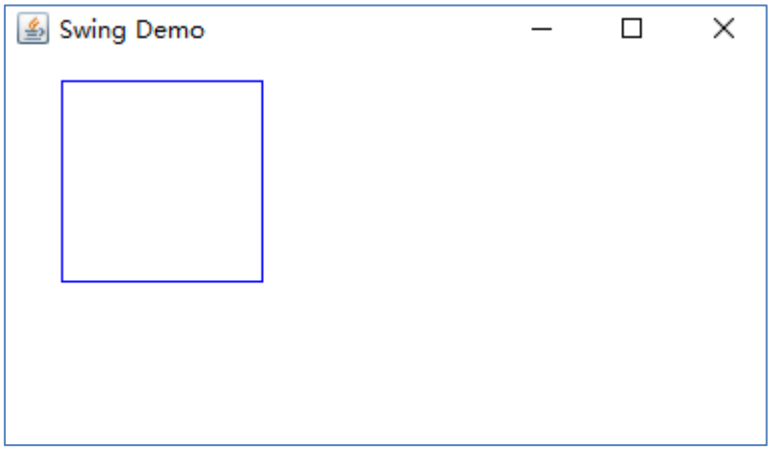
```
void drawRect(int x, int y, int width, int height)
void fillRect(int x, int y, int width, int height)
void drawOval(int x, int y, int width, int height)
void fillOval(int x, int y, int width, int height)
void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

其中，draw 打头的方法用于对形状描边，而 fill 则是颜色填充。下面以矩形绘制为例，来演示 draw 与 fill 的区别。

先看一下 drawRect()的用法，示例代码如下。

```
g.setColor(Color.blue);
g.drawRect(30,10, 100, 100); // 描边
```

运行程序，效果如下所示。



可以看到，绘制出来了一个矩形形状，位置为(30,10)，宽为 100，高为 100。这种绘制是描边的效果。

再看一下 fillRect()的用法，示例代码如下。

```
g.setColor(Color.blue);
g.fillRect(30,10, 100, 100); // 填充
```

运行程序，效果如下所示。

可以看到，fillRect()的效果是颜色填充，填充了一个蓝色矩形区域。

9.4 （练习）正弦曲线

略。请对照视频练习。

9.5 （练习）正弦曲线控制

略。请对照视频练习。

第 10 章　图片的绘制

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

10.1　绘制图片

下面讨论怎么在一个自定义的控件里绘制图片。

10.1.1　图片的加载

在 Swing 里，与图片相关的类有 Image 和 BufferedImage。其中 Image 是抽象类，表示图片的一般性质。而真实使用的是 BufferedImage 类型。

要想绘制图片，首先得准备好图片文件。在 Swing 里支持的静态图片格式为 jpg/jpeg/png 这些格式。把图片拷贝到项目里，可以作为资源文件，也可以作为本地文件。

Swing 提供了一个工具类 ImageIO，用于加载静态图片。无论是资源文件、还是本地图片，都可以用它来加载。看一下文档里的对它的说明。

```
static BufferedImage read(File input)
static BufferedImage read(InputStream input)
static BufferedImage read(URL input)
```

可以看到，ImageIO 提供若干静态的 read()方法，可以从 File, InputStream 或 URL 中来读取一个图片，返回一个 BufferedImage 对象。

例如，

```
BufferedImage image = ImageIO.read("/images/im_j20.jpg");
```

用于加载一张资源文件，一个放在 src/images/ 包下面的、名字为 im_j20.jpg 的图片，返回的 BufferedImage 即可以用于绘制。

10.1.2 图片的绘制

有了 Image/BufferedImage 对象之后，就可以绘制在控件里。查看官方文档，找到 Graphics 类里关于 drawImage()方法的说明，罗列如下。

drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)

drawImage(Image img, int x, int y, ImageObserver observer)

drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)

drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)

drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, Color bgcolor, ImageObserver observer)

drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)

可以看到它重载了多个版本的 drawImage()方法，这些方法的参数和使用是类似的。下面演示使用的是其中的一个方法，

drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)

其中，

- img，表示要绘制的图片对象
- x,y ,width, height，表示要绘制的区域
- observer，一般不管这个参数，设为 null 即可。（由于绘制过程可能耗费一定时间，这个通知器可以在绘制完成的时候被调用）

下面看一个演示，示例代码如下。

```
protected void paintComponent(Graphics g)
{
    int width = this.getWidth();
    int height = this.getHeight();
    g.clearRect(0, 0, width, height);
    try {
        URL url = getClass().getResource("/images/im_j20.jpg");
        BufferedImage image = ImageIO.read(url);
        g.drawImage(image, 0, 0, width, height,    null);
```

```
        }catch(Exception e)
        {
            e.printStackTrace();
        }
    }
```

其中，使用 ImageIO.read(url) 来加载了一个图片，紧接着调用 Graphics.drawImage()来绘制这张图片。

运行示例，效果显示如下。

由于 drawImage()里指定的绘制区域是 (0, 0, width, height)，所以看到的图片是占满了整个控件。

10.1.3 绘制的区域

下面讨论一下绘制区域的指定，示例如下。

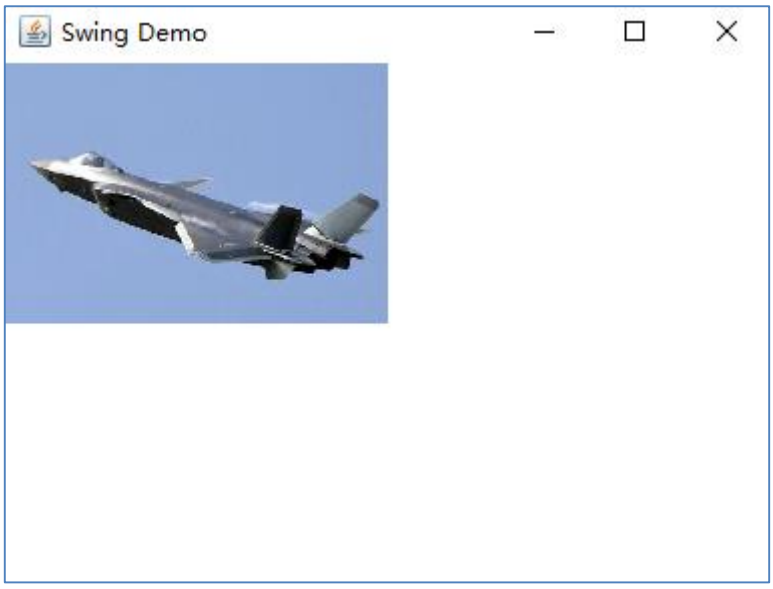
```
g.drawImage(image, 0, 0, width, height,  null);
```

其中，指定了坐标是相对于控件的坐标，（0，0）表示控件的左上角位置。在绘制的时候，会自动实现图片的缩放显示。

比如，可以在控件的左上角显示图片，占据左上的 1/4，示例如下。

```
g.drawImage(image, 0, 0, width/2, height/2,    null);
```

运行程序，效果如下所示。



由于是缩放显示，并没有保持图片的原始比例。所以在窗口大小改变时，可能显示为畸形。示例效果如下。

图中，窗口本身并拉成细长形状，此时图片仍然能够显示，并占据左上角。但是有一个明显的问题，它显示的长宽比不对，不符合图片原始的长宽比。

在 Swing 里，与图片相关的类有 Image 和 BufferedImage。其中 Image 是抽象类，表示图片的一般性质。而真实使用的是 BufferedImage 类型。

要想绘制图片，首先得准备好图片文件。在 Swing 里支持的静态图片格式为 jpg/jpeg/png 这些格式。把图片拷贝到项目里，可以作为资源文件，也可以作为本地文件。

10.1.4 图片加载的优化

一个控件可能被会多次重绘的。比如，在用户拖动窗口大小时，由于整个窗口的大小发生改变，所有的控件会被重新布局和绘制。从而，控件的 paintComponent()方法在每次重绘时被调用。

在每次控件重新绘制时，要绘制的图片对象并没有发生变化。所以一般地说，要把图片对象作为属性，在构造方法里加载。代码结构如下，

```
public class MyPanel extends JPanel
{
    private Image image ;

    public MyPanel()
    {
        try {       // 在构造方法里加载图片，仅加载一次
            URL url = getClass().getResource("/images/im_j20.jpg");
            BufferedImage image = ImageIO.read(url);
        }catch(Exception e){}
    }
    @Override
    protected void paintComponent(Graphics g)
    {
        。。。略。。。
        g.drawImage(this.image, 0, 0, w,h , nul) ;
    }
}
```

也就是在每次 paintComponent()重绘时，不需要再加载一次图片了。

10.2 锁定长宽比

下面讨论图片长宽比的问题。图片自己有自己的宽度和高度，一般在显示图片的时候，可以缩放显示，但是在缩放时图片的长宽比应该是固定的。称为“锁定长宽比”。

这也就意味着，在绘制之前应该计算一下矩形的位置。

已经条件如下：

- 控件的宽度 width，高度 height
- 图片的宽度 imgW，高度 imgH

要求计算出绘制的目标矩形的位置。示例代码如下。

```
int imgW = image.getWidth(null);
int imgH = image.getHeight(null);
// 按长宽比进行计算
int fitW = width;
int fitH = width * imgH / imgW;
if( fitH > height )
{
    fitH = height;
    fitW = height * imgW / imgH;
}
int x = (width - fitW ) /2;
int y = (height - fitH ) /2;
g.drawImage(image, x, y, fitW, fitH, null);
```

也就是说，最终计算出来的显示宽度 fitW 和显示高度 fitH，它们的比例和原始图片的宽高比是一致的。即 fitW / fitH 与 imgW / imgH 的比例相同。

该算法核心思想是：当控件比图片更扁时，以控件高度为基准进行计算。当控件比图片更窄时，时控件宽度为基准进行计算。

10.3 图片缩放工具

为了重用缩放算法，将相关算法封装为一个工具类 AfImageScaler，里面支持三种缩放方式。

(1) fitXY ，指图片拉伸占满整个控件

(2) fitCenter，指图片尽量占满控件，但是保持长宽比

(3) fitCenterInside，指图片始终保持长宽比，如果超出控件大小则实施缩放，如果控件足够显示则在中央显示原图。

它的使用方法如下所示，

```
int imgW = image.getWidth(null);
int imgH = image.getHeight(null);

// 使用  AfImageScaler 来计算
AfImageScaler scaler = new AfImageScaler(imgW, imgH, width,height);
Rectangle fit = scaler.fitCenter();
g.drawImage(image, fit.x, fit.y, fit.width, fit.height,     null);
```

其中，

- width, height 表示控件的大小
- imgW, imgH 表示图片的大小
- fit 表示经过计算后的，最终的显示位置（一个 Rectangle 区域）

10.4 图片显示控件

提供了工具类 AfImageView，用于显示一个图片。支持缩放显示。

示例代码如下：

```
AfImageView imageView = new AfImageView();
root.add(imageView, BorderLayout.CENTER);
// 设置缩放类型
imageView.setScaleType(AfImageView.FIT_CENTER_INSIDE);
// 设置要显示的图片
Image image = imageFromResource("/images/im_j20.jpg");
imageView.setImage(image );
```

具体讲解与演示，请对照视频教程。

10.5 （练习）背景图片

略。请对照视频练习。

第 11 章　图片的绘制

作者：邵发
官网：http://afanihao.cn
QQ 群：　495734195
本教程是 Java 学习指南系列的 Swing 入门篇（视频教程）的配套学习资料，最新版本请到官网下载。

11.1　鼠标事件

本章讨论鼠标事件的处理。鼠标事件（MouseEvent），包括以下几种类型：

- 鼠标点击 mouseClicked
- 鼠标按下 mousePressed
- 鼠标抬起 mouseReleased
- 鼠标移入 mouseEntered
- 鼠标移出 mouseExited
- 鼠标移动 mouseMoved
- 鼠标拖动 mouseDragged
- 鼠标滚轮 mouseWheelMoved

所有的 JComponent 都支持三个鼠标相关的监听器，依次为：

(1) addMouseListener()：　点击、按下、抬起、移入、移出

(2) addMouseMotionListener()：　移动、拖动

(3) addMouseWheelListener() ：　鼠标滚轮转动

举例来说，如果想要监听鼠标点击事件，应该调用 addMouseListener()。如果想要监听鼠标移动事件，则调用 addMouseMotionListener()。

下面，给一个 JPanel 控件添加鼠标事件处理，使用匿名类的写法，示例代码如下。

```
panel.addMouseListener(new MouseListener() {
    @Override
    public void mouseReleased(MouseEvent e)
    {
```

```
            }
            @Override
            public void mousePressed(MouseEvent e)
            {
            }
            @Override
            public void mouseExited(MouseEvent e)
            {
            }
            @Override
            public void mouseEntered(MouseEvent e)
            {
            }
            @Override
            public void mouseClicked(MouseEvent e)
            {
            }
        });
```

可以看到，MouseListener 是一个接口，里面定义了 5 个抽象方法，依次为：mousePressed 按下、mouseReleased 抬起、mouseClicked 点击、mouseEntered 进入、mouseExited 移出。

其中，mouseClicked 事件其实和就是 mousePressed 和 mouseRelease 事件的组合，因为一次点击事件必然是先按下、后抬起，而且按下和抬起的位置应该是同位置。

每一个事件处理回调中，都传递了一个 MouseEvent 对象作为参数，此对象里携带了刚刚发生的鼠标事件的具体信息。在 MouseEvent 类里提供了若干方法来获取这些信息。罗列如下，

- getX() / getY() ：点击中的坐标，相对于该控件
- getXOnScreen() / getOnScreen() ：相对于屏幕的坐标
- getSource()：事件源，即点中的控件
- getButton()： 左键、中键、右键
- getClickCount()：单击、双击、三击

在事件处理回调中，可以从 MouseEvent 中取得这些信息，并根据这些信息做

出相应的处理。示例代码如下，

```
@Override
public void mouseClicked(MouseEvent e)
{
    if(e.getButton() == MouseEvent.BUTTON1)
        System.out.println("左键");
    else if(e.getButton() == MouseEvent.BUTTON2)
        System.out.println("中键");
    else if(e.getButton() == MouseEvent.BUTTON3)
        System.out.println("右键");

    int cc = e.getClickCount();
    if(cc == 1)
        System.out.println("单击");
    else if(cc == 2)
        System.out.println("双击");

    System.out.println("鼠标位置: " + e.getX() + "," + e.getY());
}
```

在此例中，演示了如何从 MouseEvent 中取出鼠标事件信息。在实际应用中，可以根据点击的位置、按钮，做出相应的处理。

11.2 鼠标适配器

在上一节的例子中，要添加一个鼠标事件的处理，需要重写 MouseListener 里规定的 5 个方法。但实际上，我们只对 MouseListener 里面的一个方法感兴趣，其余四个方法都是置空的。

这么写的话，在形式上有了一个冗余。我们希望，在重写 MouseListener 的时候，只实现自己关心的事件回调。怎么解决呢？

在 Swing 里提供了一个 MouseAdapter 的类，可以解决这个问题。看一个 MouseAdapter 类的实现（此类在 java.awt.event.*包下面）。

```
public abstract class MouseAdapter implements MouseListener,
                MouseWheelListener, MouseMotionListener
```

```
{
    public void mouseClicked(MouseEvent e) {}
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mouseWheelMoved(MouseWheelEvent e){}
    public void mouseDragged(MouseEvent e){}
    public void mouseMoved(MouseEvent e){}
}
```

可以发现，这个 MouseAdapter 其实只是一个空架子，表面上它是实现了 MouseListener、MouseWheelListener 和 MouseMotionListener 这三个接口，但实质上它就是把每个方法直接置空了。

回顾一下第 6 章里，AfSimpleLayout 的设计思想是不是和它一样呢？

有了这个 MouseAdapter 类，在创建鼠标事件监听器时就会简洁一下，只把我们关心的事件回调重写一下就可以了。示例代码如下。

```
panel.addMouseListener( new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e)
    {
        。。。略。。。
    }
});
```

在语法上，addMouseListener()要求传入一个 MouseListener 的子类，而 MouseAdapter 刚好符合要求。

此时我们只关心鼠标点击事件，所以只需要重写 mouseClicked()方法就行了，其他几个方法不需要写，不存在语法错误。

11.3 （练习）图片查看器

略。请对照视频练习。

11.4 （练习）手绘自由曲线

略。请对照视频练习。