

## swing 入门教程

swing 简介(2009-04-20 23:52:40)

### UI 组件简介

在开始学习 **Swing** 之前，必须回答针对真正初学者的问题：什么是 **UI**？初学者的答案是“用户界面”。但是因为本教程的目标是要保证您不再只是个初学者，所以我们需要比这个定义更高级的定义。

所以，我再次提出这个问题：什么是 **UI**？您可能把它定义成您按下的按钮、打字的地址栏、打开和关闭的窗口，等等，这些都是 **UI** 的元素，但是除了在屏幕上看到的这些之外，还有更多都是 **UI** 元素。比如鼠标、键盘、音量、屏幕颜色、使用的字体，以及一个对象相对于另一个对象的位置，这些都包含在 **UI** 之中。实际上，在计算机和用户的交互之中扮演角色的任何对象都是 **UI** 的组成部分。这看起来足够简单，但是您应当惊讶的是，有许多人和大型公司已经为它努力了很多年。实际上，现在有的大学专业的惟一课程就是研究这种交互。

### Swing 的角色

**Swing** 是 Java 平台的 **UI** —— 它充当处理用户和计算机之间全部交互的软件。它实际上充当用户和计算机内部之间的中间人。**Swing** 到底是如何做这项工作的呢？它提供了处理前面一节中描述的 **UI** 各方面内容的机制：

- 键盘：**Swing** 提供了捕捉用户输入的方法。
- 颜色：**Swing** 提供改变在屏幕上看到的颜色的方法。
- 打字的地址栏：**Swing** 提供了文本组件，处理所有普通任务。

- 音量：Swing 不太擅长。

无论如何，Swing 为您提供了创建自己的 UI 所需要的所有工具

## MVC

Swing 甚至走得更远一步，在基本的 UI 原则之上又放上了一个公共的设计模式。这个设计模式叫做模型-视图-控制器（Model-View-Controller，MVC），它试图“把角色分开”。MVC 让负责显示的代码、处理数据的代码、对交互进行响应并驱动变化的代码彼此分离。

有点迷惑？如果我为这个设计模式提供一个现实世界的非技术性示例，它就比较容易了。请想像一次时装秀。把秀场当成 UI，假设服装就是数据，是展示给用户的计算机信息。现在，假设这次时装秀中只有一个人。这个人设计服装、修改服装、同时还在 T 台上展示这些服装。这看起来可不是一个构造良好的或有效率的设计。

现在，假设同样的时装秀采用 MVC 设计模式。这次不是一个人做每件事，而是将角色分开。时装模特（不要与 MVC 缩写中的模型混淆）展示服装。他们扮演的角色是视图。他们知道展示服装（数据的）适当方法，但是根本不知道如何创建或设计服装。另一方面，时装设计师充当控制器。时装设计师对于如何在 T 台上走秀没有概念，但他能创建和操纵服装。时装模特和设计师都能独立地处理服装，但都有自己的专业领域。

这就是 MVC 设计模式背后的概念：让 UI 的每个方面处理它擅长的工作。如果您仍然不明白，那么教程后面的示例有望消除您的迷惑 —— 但是在您继续进行的时候，请记住基本的原则：用可视组件显示数据，同时让其他类操纵数据。

## JComponent

Swing 的整个可视组件库的基础构造块是 JComponent。它是所有组件的父类。它是一个抽象类，所以不能创建 JComponent，但是作为类层次结构的结果，从字面意义来说它包含了数百个函数，Swing 中的每个组件都可以使用这些函数。显然，有些概念要比其他概念重要，所以对于本教程，需要学习的重要的东西是：

- JComponent 不仅是 Swing 组件的基类，还是定制组件的基类（有关的更多信息在“[中级 Swing](#)”教程中）。
- 它为所有组件提供了绘制的基础架构 — 一些方便进行组件定制的东西（同样，在“[中级 Swing](#)”中，有关于这个主题的更多信息）。
- 它知道如何处理所有的键盘按键。所以类只需要侦听特定的键。
- 它包含 add() 方法，可以添加其他 JComponent。换种方式来看，可以把任意 Swing 组件添加到其他任何 Swing 组件，从而构造嵌套组件（例如，JPanel 包含 JButton，甚至包含一些古怪的组合，例如 JMenu 包含 JButton）。

## 简单的 swing 小部件

### JLabel

Swing 库中最基础的组件是 JLabel。它所做的正是您所期望的：呆在那儿，看起来很漂亮，描述其他组件。下图显示了的 JLabel 实际应用：

## JLabel

不太吸引人，但是仍然有用。实际上，在整个应用程序中，不仅把 JLabel 用作文本描述，还将它用作图片描述。每当在 Swing 应用程序中看到图片的时候，它就有可能是 JLabel。JLabel 对于 Swing 初学者来说没有许多意料之外的方法。基本的方法包括设置文本、图片、对齐以及标签描述的其他组件：

- `get/setText()`: 获取/设置标签的文本。
- `get/setIcon()`: 获取/设置标签的图片。
- `get/setHorizontalAlignment()`: 获取/设置文本的水平位置。
- `get/setVerticalAlignment()`: 获取/设置文本的垂直位置。
- `get/setDisplayedMnemonic()`: 获取/设置标签的访问键（下划线文字）。
- `get/setLabelFor()`: 获取/设置这个标签附着的组件，所以当用户按下 `Alt+` 访问键时，焦点转移到指定的组件。

## JButton

Swing 中的基本动作组件 JButton，是与每个窗口中都能看到的 OK 和 Cancel 一样的按钮；这些按钮所做的正是您希望它们做的工作 — 在单击它们之后，将发生一些事情。到底会发生什么呢？您必须定义发生的内容（请参阅 [事件](#)，以获得更多信息）。一个 JButton 实例看起来如下所示：

## JButton

用来改变 `JButton` 属性的方法与 `JLabel` 的方法类似（您可能发现，在大多数 `Swing` 组件中，这些属性都类似）。它们控制文本、图片和方向：

- `get/setText()`: 获取/设置标签的文本。
- `get/setIcon()`: 获取/设置标签的图片。
- `get/setHorizontalAlignment()`: 获取/设置文本的水平位置。
- `get/setVerticalAlignment()`: 获取/设置文本的垂直位置。
- `get/setDisplayedMnemonic()`: 获取/设置访问键（下划线字符），与 `Alt` 按钮组合时，造成按钮单击。

除了这些方法，我还要介绍 `JButton` 包含的另外一组方法。这些方法利用了按钮的所有不同状态。状态是对组件进行描述的一个属性，通常采用真/假设置。在 `JButton` 中，可以包含以下可能状态：活动/不活动、选中/没选中、鼠标经过/鼠标离开、按下/没按下，等等。

另外，可以组合这些状态，例如按钮可以在鼠标经过的同时被选中。现在您可能会问自己用这些状态到底要做什么。作为示例，请看看您的浏览器上的后退按钮。请注意在鼠标经过它的时候，图片是如何变化的，在按下该按钮时，图片又是如何变化的。这个按钮利用了不同的状态。每个状态采用不同的图片，这是提示用户交互正在进行的一种普遍并且有效的方式。

`JButton` 上的状态方法是：

- `get/setDisabledIcon()`
- `get/setDisableSelectedIcon()`
- `get/setIcon()`
- `get/setPressedIcon()`
- `get/setRolloverIcon()`
- `get/setRolloverSelectedIcon()`
- `get/setSelectedIcon()`

## JTextField

Swing 中的基本文本组件是 `JTextField`，它允许用户在 UI 中输入文本。我肯定您熟悉文本字段：要掌握本教程，则必须使用一个文本字段输入用户名和口令。您输入文本、删除文本、选中文本、把文字四处移动 — Swing 替您负责所有这些工作。作为 UI 开发人员，利用 `JTextField` 时，实际上并不需要做什么。

在任何情况下，这是 `JTextField` 实际使用时看起来的样子：

## JTextField

在处理 `JTextField` 时，只需要关注一个方法 — 这应当是很明显的，这个方法就是设置文本的方法：`getText()`，用于获取/设置 `JTextField` 中的文本。

## JFrame

迄今为止，我介绍了 Swing 的三个基本构造块：标签、按钮和文本字段；但是现在需要个地方放它们，希望用户知道如何处理它们。`JFrame` 类就是做这个的——它是一个容器，允

许您把其他组件添加到它里面，把它们组织起来，并把它们呈现给用户。它有许多其他好处，但是我认为先看看它的图片最简单：

## JFrame

**JFrame** 实际上不仅仅让您把组件放入其中并呈现给用户。比起它表面上的简单性，它实际上是 **Swing** 包中最复杂的组件。为了最大程度地简化组件，在独立于操作系统的 **Swing** 组件与实际运行这些组件的操作系统之间，**JFrame** 起着桥梁的作用。**JFrame** 在本机操作系统中是以窗口的形式注册的，这么做之后，就可以得到许多熟悉的操作系统窗口的特性：最小化/最大化、改变大小、移动。但是对于本教程的目标来说，把 **JFrame** 当作放置组件的调色板就足够了。可以在 **JFrame** 上调用的一些修改属性的方法是：

- `get/setTitle()`: 获取/设置帧的标题。
- `get/setState()`: 获取/设置帧的最小化、最大化等状态。
- `is/setVisible()`: 获取/设置帧的可视状态，换句话说，是否在屏幕上显示。
- `get/setLocation()`: 获取/设置帧在屏幕上应当出现的位置。
- `get/setSize()`: 获取/设置帧的大小。
- `add()`: 将组件添加到帧中。

## 简单应用程序

就像所有的“x 入门”教程一样，本教程也包含必不可少的 HelloWorld 演示。但这个示例不仅对观察 Swing 应用程序如何工作有用，还对确保设置正确很有用。一旦使这个简单的应用程序能够成功运行，那么之后的每个示例也将能够运行。下图显示了完成后的示例：

## HelloWorld 示例

第一步是创建类。将组件放在 JFrame 上的 Swing 应用程序需要继承 JFrame 类，如下所示：

```
public class HelloWorld extends JFrame
```

这样做之后，就得到上面描述的所有 JFrame 属性，最重要的是操作系统对窗口的本机支持。下一步是把组件放在屏幕上。在这个示例中，使用了一个 null 布局。在教程的后面部分，您将学到更多关于布局和布局管理器的内容。但对于这个示例，可以用数字表示 JFrame 上的像素位置：

```
public HelloWorld()  
{  
    super();  
    this.setSize(300, 200);  
    this.getContentPane().setLayout(null);  
    this.add(getJLabel(), null);  
    this.add(getJTextField(), null);  
    this.add(getJButton(), null);  
}
```



```
        this.setTitle("HelloWorld");  
    }  
}
```

```
private javax.swing.JLabel getJLabel() {  
    if(jLabel == null) {  
        jLabel = new javax.swing.JLabel();  
        jLabel.setBounds(34, 49, 53, 18);  
        jLabel.setText("Name:");  
    }  
    return jLabel;  
}
```

```
private javax.swing.JTextField getJTextField() {  
    if(jTextField == null) {  
        jTextField = new javax.swing.JTextField();  
        jTextField.setBounds(96, 49, 160, 20);  
    }  
    return jTextField;  
}
```

```
private javax.swing.JButton getJButton() {  
    if(jButton == null) {
```

```
jButton = new javax.swing.JButton();

jButton.setBounds(103, 110, 71, 27);

jButton.setText("OK");

}

return jButton;

}
```

现在组件都放在了 **JFrame** 上，并且需要在屏幕上显示 **JFrame**，并让应用程序可以运行。

就像在所有的 Java 应用程序中一样，必须添加一个 **main** 方法，才能让 **Swing** 应用程序运行。在这个 **main** 方法中，只需要创建 **HelloWorld** 应用程序对象，然后调用其 **setVisible()** 即可：

```
public static void main(String[] args)

{

    HelloWorld w = new HelloWorld();

    w.setVisible(true);

}
```

完成了！这就是创建应用程序的所有过程。

完整代码如下：

```
package cn.edu.jnu.www;

import javax.swing.*;

import javax.swing.event.*;

import java.awt.*;

import java.awt.event.*;

public class HelloWorld extends JFrame{

    private JLabel jLabel;

    private JTextField jTextField;

    private JButton jButton;

    public HelloWorld()

    {

        super();

        this.setSize(300, 200);

        this.getContentPane().setLayout(null);

        this.add(getJLabel(), null);

        this.add(getJTextField(), null);

        this.add(getJButton(), null);

        this.setTitle("HelloWorld");

    }
```

```
private javax.swing.JLabel getJLabel () {  
  
    if(jLabel == null) {  
  
        jLabel = new javax.swing.JLabel ();  
  
        jLabel.setBounds(34, 49, 53, 18);  
  
        jLabel.setText("Name:");  
  
    }  
  
    return jLabel;  
  
}
```

```
private javax.swing.JTextField getJTextField() {  
  
    if(jTextField == null) {  
  
        jTextField = new javax.swing.JTextField();  
  
        jTextField.setBounds(96, 49, 160, 20);  
  
    }  
  
    return jTextField;  
  
}
```

```
private javax.swing.JButton getJButton() {  
  
    if(jButton == null) {  
  
        jButton = new javax.swing.JButton();  
  
        jButton.setBounds(103, 110, 71, 27);  
  
        jButton.setText("OK");  
  
    }  
  
}
```

```
        return jButton;  
    }  
  
    public static void main(String[] args)  
    {  
        HelloWorld w = new HelloWorld();  
        w.setVisible(true);  
    }  
}
```

附加的 swing 小部件(上)

## JComboBox

在这一节中，我将介绍 **Swing** 库中的其他全部组件、如何使用它们、它们看起来是什么样子的，等等，这部分内容应当让您更好地了解 **Swing** 为 **UI** 开发人员提供了什么。

我们从 **JComboBox** 开始介绍。组合框与下拉选择相似，区别在于使用组合框时用户可以不从列表中选择项目，还可以选择一个（而且只有一个）项目。在某些版本的组合框中，还可以输入自己的选择。浏览器的地址栏就是一个示例：它是一个允许输入自己选项的组合框。

以下是 **JComboBox** 在 **Swing** 中看起来的样子：

## JComboBox

JComboBox 的重要函数包括 JComboBox 包含的数据。需要有一种方法来设置

JComboBox 中的数据、修改数据、在用户选择时得到用户的选择。可以使用以下

JComboBox 方法：

- `addItem()`：添加一个项目到 JComboBox.
- `get/setSelectedIndex()`：获取/设置 JComboBox 中选中项目的索引。
- `get/setSelectedItem()`：获取/设置选中的对象。
- `removeAllItems()`：从 JComboBox 删除所有对象。
- `removeItem()`：从 JComboBox 删除特定对象。

## JTextField

JTextField 的一个细微变化是 JPasswordField，它允许您隐藏在文本字段区域中显示的字符。毕竟，在您输入口令的时候，如果每个人都能看到，那可没什么好处？可能根本就不好，而且在私人数据如此脆弱的今天，您需要所有能够得到的帮助。以下是 JPasswordField 在 Swing 中看起来的样子：

### JPasswordField

JPasswordField 上额外的“安全性”方法对 JTextField 的行为做了轻微改变，所以不能阅读文本：

- `get/setEchoChar()`：获取/设置每次字符输入时在 JPasswordField 中显示的字符。在获取口令时，不会返回“回声”，而是返回实际的字符。

- `getText()`: 不应当使用这个函数，因为它会带来可能的安全问题（String 会保存在内存中，可能的堆栈转储会暴露口令）。
- `getPassword()`: 这是从 `JPasswordField` 中获得口令的恰当方法，因为它返回一个包含口令的 `char[]`。为了保证恰当的安全性，数组应当被清为 0，以确保它不会保留在内存中。

## JCheckBox/JRadioButton

`JCheckBox` 和 `JRadioButton` 组件向用户呈现选项，通常采用多选的格式。区别是什么？

从实践的角度来说，它们没有那么不同。它们的行为方式相同。但是，在一般的 UI 实践中，它们有细微差异：`JRadioButton` 通常组合在一起，向用户呈现带有必选答案的问题，而且这些答案具有强制性（这意味着问题只能有一个答案）。`JRadioButton` 的行为保证了这个用法。一旦选择了 `JRadioButton`，就不能取消对它的选择，除非选择了在同一组中的另外一个单选钮。从效果上看，这就保证了选项的惟一和必选。`JCheckBox` 的不同在于，允许随机地选择/取消除选择，并允许为问题选择多个答案。

这里是个示例。问题“您是男孩还是女孩!”有两个惟一答案选项“男孩”或“女孩”。用户必须选择一个，不能同时选中。另一方面，问题“您的习惯是什么？”的答案有“跑步”、“睡觉”或“阅读”，不应当只允许为此问题选择一个答案，因为人们可能有不止一个习惯。

把这些 `JCheckBox` 或 `JRadioButton` 捆绑成一组的类是 `ButtonGroup` 类。它允许把选项组织在一起（例如“男孩”和“女孩”），这样，其中一个被选择时，另外一个就自动取消选择。

以下是 JCheckBox 和 JRadioButton 在 Swing 中看起来的样子：

## JCheckBox 和 JRadioButton

需要记住的重要的 ButtonGroup 方法是：

- `add()`：添加 JCheckBox 或 JRadioButton 到 ButtonGroup。
- `getElement()`：获得 ButtonGroup 中的全部组件，允许对它们进行迭代，找到其中选中的那个。

## JMenu/JMenuItem/JMenuBar

JMenu、JMenuItem 和 JMenuBar 组件是在 JFrame 中开发菜单系统的主要构造块。任何菜单系统的基础都是 JMenuBar。它平淡而乏味，但却是必需的，因为每个 JMenu 和 JMenuItem 都要用它构建。要用 `setJMenuBar()` 方法把 JMenuBar 附着到 JFrame。一旦将它附加到 JFrame 中，就可以添加所有想要的菜单、子菜单和菜单项。

JMenu/JMenuItem 的区别看起来可能很明显，但实际上，在幕后看起来并不像表面那样。看看类的类层次结构，就知道 JMenu 是 JMenuItem 的子类。但是，在表面上，它们是有区别的：用 JMenu 包含其他 JMenuItem 和 JMenu；JMenuItem 在选中时触发操作。

JMenuItem 也支持快捷键的概念。与您用过的大多数应用程序一样，Swing 应用程序允许您按下 Ctrl+（某个键）来触发一个操作，就像选中菜单键本身一样。想想用来剪切和粘贴的快捷键 Ctrl+X 和 Ctrl+V。

除此之外，JMenu 和 JMenuItem 都支持访问键。用 Alt 键与某个字母关联，模拟菜单本身的选择（例如，在 Windows 中按下 Alt+F，然后按下 Alt+x 就可以关闭应用程序）。



以下是包含 JMenu 和 JMenuItem 的 JMenuBar 在 Swing 中的样子：

## JMenuBar、JMenu 和 JMenuItem

这些类需要的重要方法是：

- JMenuItem and JMenu:
  - `get/setAccelerator()`：获取/设置用作快捷键的 Ctrl+ 键。
  - `get/setText()`：获取/设置菜单的文本。
  - `get/setIcon()`：获取/设置菜单使用的图片。
- JMenu 专用：
  - `add()`：添加另外一个 JMenu 或 JMenuItem 到 JMenu（创建嵌套菜单）。

## JSlider

在应用程序中 JSlider 支持数值变化。它是一种迅速而简单的方式，不仅能让用户以可视形式获得他们当前选择的反馈，还能得到可以接受的值的范围。想像一下这种情况：可以提供一个文本字段，允许用户输入值，但是这样做就带来了额外的麻烦，要确保输入的值是数字，还要确保数字符合要求的数值范围。例如，如果有一个金融 Web 站点，它向您提问要在股票上投资的百分比，那么您不得不检查在文本字段中输入的值，以确保它们是数字，而且在 0 到 100 之间。如果换用 JSlider，那么就可以确保选择的是指定范围内的数字。

在 Swing 中，JSlider 看起来如下所示：

## JSlider

JSlider 中的重要方法是：

- `get/setMinimum()`: 获取/设置可以选择的最小值。
- `get/setMaximum()`: 获取/设置可以选择的最大值。
- `get/setOrientation()`: 获取/设置 `JSlider` 是上/下还是左/右滚动条。
- `get/setValue()`: 获取/设置 `JSlider` 的初始值。

## JSlider

与 `JSlider` 非常像，可以用 `JSpinner` 允许用户选择一个整数值。`JSlider` 的一个主要优势就是比 `JSlider` 的空间紧凑。但是，它的不足就是无法方便地设置其边界。

但是，两个组件之间的比较仅此而已。`JSpinner` 更加灵活，可以用来在任意组的值之间进行选择。除了在数字间选择，它还可以用来在日期、名称、颜色和任何事之间进行选择。这使 `JSpinner` 极为强大，允许您提供其中只包含预定义的选择的组件。使用这种方式，它与 `JComboBox` 类似，但是它们的应用不能互换。只应把 `JSpinner` 用在逻辑上连续的选择——数字和日期是最合逻辑的选择。而另一方面，在呈现看起来随机的选择并且选择之间没有连接的时候，`JComboBox` 是更好的选择。

`JSpinner` 看起来如下所示：

### JSpinner

重要方法是：

- `get/setValue()`: 获取/设置 `JSpinner` 的初始值，在基本实例中，需要是整数。
- `getNextValue()`: 获取按下上箭头按钮之后应当选中的下一个值。
- `getPreviousValue()`: 获取按下下箭头按钮之后应当选中的前一个值。

## JToolBar

JToolBar 充当其他组件（JButton、JComboBox 等）的调色板，共同构成您在大多数应用程序中熟悉的工具栏。工具栏允许程序把常用的命令放在可以迅速发现的位置，并把它们以常用命令组的形式组合在一起。一般（但不总是这样）情况下，工具栏按钮在菜单栏中会有对应的命令。虽然这不是必需的，但已经变成了一种公共实践，您也应当试着这么做。

JToolBar 也提供了您在其他工具栏中看到过的其他功能：“浮动”的能力（也就是成为主帧顶部独立的帧）。

下图显示了一个非浮动 JToolBar：

### 非浮动 JToolBar

对于 JToolBar，要记住的重要方法是：`is/setFloatable()`，它获取/设置 JToolBar 是否可以浮动。

## JToolTip

您可能到处都看到过 JToolTip，但是从来不知道它们叫什么。它们就像您鞋带上的塑料部件——到处都有，但是您就是不知道它们正确的名字（如果您想知道，那么可以叫它们 金属箍）。JToolTip 就是您将鼠标停留在某个东西上面的时候弹出来的小“泡泡”。它们在应用程序中可能非常有用，可以为难用的项目提供帮助、扩展信息，甚至在拥挤的 UI 中显示某个项目的完整文本。在 Swing 中，可以通过把鼠标放在某个组件上的特定时间来触发它们；它们通常在鼠标处于不活动状态大约 1 秒钟之后显示。只要鼠标还停留在那个组件上，它们就保持可见。

JToolTip 的重要部分是它的易用性。setToolTip() 方法是 JComponent 类中的一个方法，这意味着每个 Swing 组件都能有一个与之关联的工具提示。虽然 JToolTip 本身也是一个 Swing 类，但目前，对于您的需要，它确实没有提供更多功能，而且本身也不该被创建。可以通过调用 JComponent 的 setToolTip() 函数访问和使用它。

以下是 JToolTip 看起来的样子：

## A JToolTip

附加的 swing 小部件(下)

## JOptionPane

JOptionPane 是在 Swing 中类似“快捷方式”的东西。通常，作为 UI 开发人员，您需要向用户呈现快速信息，让用户了解错误和信息。甚至可能想得到一些快速数据，例如名称或数字。在 Swing 中，JOptionPane 类为这些东西提供了快捷方式，但这并不是它必须完成的任务。不需要让每个开发人员重头开始重复相同的工作，Swing 已经提供了这个基本的但很有用的类，为 UI 开发人员提供了获取和接收简单消息的简易方法。

以下是一个 JOptionPane：

## JOptionPane

使用 JOptionPane 时有点麻烦的是可以使用的所有选项。虽然简单，但是它仍然提供了大量选项，这些选项有可能造成混淆。学习 JOptionPane 的最好方法就是使用它；编写代码，观察弹出的是什么。这个组件几乎可以让您修改它的每一方面：帧标题、消息本身、

显示的图标、按钮选项，以及是否需要文本响应。因为有太多的可能性，无法在本教程中一一列举它们，所以您最好是访问 `JOptionPane` 的 [API](#) 页面，查看它的众多可能性。

## JTextArea

`JTextArea` 比 `JTextField` 更进了一步。`JTextField` 被局限在一行文本中，而 `JTextArea` 扩展了这个能力，支持多行文本。可以把它想像成一个空白页，您可以在其中的任意地方进行输入。正如您可能猜到的，`JTextArea` 包含许多与 `JTextField` 相同的功能，毕竟，它们实际上是相同的组件。但是 `JTextArea` 提供了一些额外的重要功能，可以把它区别开。这些功能包括单词自动换行的能力（即把长文本自动换行到下一行，而不是将单词从中断开）、对文本自动换行的能力（即把长的文本行移动到下一行，而不是创建一个需要水平滚动条的非常长的行）。

Swing 中的 `JTextArea` 看起来就像您期望的那样：

### A JTextArea

支持行和单词的自动换行的重要方法是：

- `is/setLineWrap()`：设置在行过长的时候是否要自动换行。
- `is/setWrapStyleWord()`：设置在单词过长的时候是否要把长单词移到下一行。

## JScrollPane

上面的示例构造完成之后，假设 `JTextArea` 包含太多文本，而给定的空间中容纳不下，那这该怎么办？如果您以为会自动出现滚动条，那么很不幸，您错了。`JScrollPane` 添补了这

个空白，为 **Swing** 组件提供了处理所有与滚动条相关的动作。所以虽然为每个需要的组件提供滚动块可能有些痛苦，但是一旦添加了它，它就会自动处理每件事，包括在需要的时候隐藏/显示滚动条。

除了用需要自动换行的组件创建 **JScrollPane** 之外，不必直接处理它。根据上面的示例，用 **JTextArea** 调用 **JScrollPane** 的构造函数，这为 **JTextArea** 创建了文本过长时滚动的能力：

```
JScrollPane scroll = new JScrollPane(getTextArea()); add(scroll);
```

更新后的示例看起来如下所示：

## **JScrollPane 示例**

**JScrollPane** 也公开了它将创建的两个 **JScrollBar**。这些 **JScrollBar** 组件也包含一些方法，可以用这些方法来修改组件的行为（虽然它们不在本教程的范围之内）。

使用 **JScrollPane** 需要的方法是：

- **getHorizontalScrollBar()**：返回水平的 **JScrollBar** 组件。
- **getVerticalScrollBar()**：返回垂直的 **JScrollBar** 组件。
- **get/setHorizontalScrollBarPolicy()**：这个“策略”可以是以下三个之一：**Always**、**Never** 或 **As Needed**。
- **get/setVerticalScrollBarPolicy()**：与水平函数相同。

## **JList**

JList 是一个有用的组件，用于向用户呈现许多选择。可以把它当作 JComboBox 的扩展。

JList 提供了更多选择，并添加了多选的能力。在 JList 与 JComboBox 之间进行选择通常取决于以下两个特性：如果需要多选，或者选择的选项超过 15 个（虽然这个数字并不是通用规则），那么就应当选择 JList。

应用将 JList 与 JScrollPane 结合使用，就像上面演示的那样，因为它能够呈现比它的空间所能容纳的更多的选项。

JList 包含选择模型的概念（在 JTable 中也会看到），在这里，可以设置 JList 接受不同类型的选择。这些类型是：单一选择（只能选择一项）、单一间隔选择（只能选择相邻选项），以及任意多项或者多项间隔选择（可以选择任意数量、任意组合的选择）。

JList 是第一个我称为“复杂组件”的组件，该复杂组件还包含 JTable 和 JTree，它们支持大量的定制变化，其中包括改变 UI 的表现方式、处理数据的方式。因为本教程只是想介绍基础知识，所以我不想深入这些更高级的功能，但是在使用这些组件时有件事需要记住——它们带来的挑战要比目前为止介绍过的所有组件都大。

JList 在 Swing 中看起来如下所示：

JList

JList 中有许多处理数据的函数，而且根据我的说法，这些也只不过是使用 JList 的细节的皮毛而已。以下是一些基本方法：

- `get/setSelectedIndex()`: 获取/设置列表中选中的行；在多选择列表的情况下，返回一个 `int[]`。
- `get/setSelectionMode()`: 与上面解释的一样，获取/设置选择模式，模式有：单一、单一间隔和多选间隔。
- `setListData()`: 设置在 `JList` 中使用的数据。
- `get/setSelectedValue()`: 获得选中的对象（与选中行号对应）。

## JTable

在考虑 `JTable` 时，请想像一下一个 Excel 工作表，这样就可以对 `JTable` 在 `Swing` 中的作用有一个清晰的印象。它与工作表共享许多相同的特征：单元格、行、列、移动列、隐藏列等。`JTable` 把 `JList` 的想法更进了一步。它不是在一列中显示数据，而是在多列中显示数据。让我们以人为例。`JList` 只能显示人的一个属性 —— 例如他或她的名字。而 `JTable` 就能够显示多个属性 —— 名字、年龄、地址，等等。`JTable` 是支持提供数据的大多数信息的 `Swing` 组件。

不幸的是，作为代价，`JTable` 也是最难对付的 `Swing` 组件。许多 UI 开发人员都为了学习 `JTable` 的每个细节而头痛。在这里，我希望我把您解救出来，只用您的 `JTable` 知识处理问题。

许多 `JList` 中的概念也扩展到了 `JTable`，其中包括不同的选择间隔的概念。但是 `JList` 中一列的概念变成了 `JTable` 的单元格的概念。这意味着在 `JTable` 中进行选择时会有不同的方式，例如列、行或者一个单元格。

在 `Swing` 中，`JTable` 看起来如下所示：



## JTable

最后，JTable 的大多数函数都超出本教程的范围；“中级 Swing”会深入这个复杂组件的更多细节。

## JTree

JTree 是另外一个复杂组件，它不像 JTable 那样难用，但是也不像 JList 那么容易。使用 JTree 时麻烦的部分是它要求的数据模型。

JTree 的功能来自树的概念，树有分支和叶子。您在 Windows 中使用 IE 浏览器时，可能非常熟悉这个概念 —— 可以展开和折叠分支，显示可以选择和取消选择的不同叶子。

您很有可能发现树在应用程序中不像表格或列表那样有用，所以在 Internet 上没有许多有帮助的这方面的示例。实际上，像 JTable 一样，JTree 没有什么入门级的功能。如果决定使用 JTree，那么立即就可以达到中级水平，当然还必须学习随之而来的概念。因此，示例应用程序没有介绍 JTree，所以也很不幸，不管是入门教程还是中级教程，都没有涉及这个不太流行的组件。

但是，树有一些时候是符合需求的合理的 UI 组件。文件/目录系统就是一个示例（就像在 IE 浏览器中那样），而且当数据采取层次结构的时候，也就是说数据采用树的形式的时候，JTree 就是最佳组件。

在 Swing 中，JTree 看起来如下所示：

## JTree

## Swing 概念

### 布局、模型和事件

既然您已经知道了大多数（肯定不是全部）可以用来制作 UI 的组件，那么就必须实际使用它们做些什么。您不能只是随意地把它们放在屏幕上，然后就指望它们立即就能工作。您必须把它们放在特定的点上，对它们的交互作出反应，然后根据交互更新它们，用数据填充它们。要填满 UI 知识的这片空白，还需要更多地学习 UI 的其他重要部分。

所以，让我们来研究以下内容：

- **布局：**Swing 包括许多布局，布局也是类，负责处理组件在应用程序中的摆放位置，以及在应用程序改变尺寸或者删除、添加组件时对组件进行相应处理。
- **事件：**您需要对按下按钮、单击鼠标和用户在 UI 上能做的每件事进行响应。想像一下，如果不能响应会发生什么 —— 用户单击之后，什么变化也没有。

- **模型：** 对于更高级的组件（列表、表格和树），以及一些像 JComboBox 这样的更容易的组件来说，模型是处理数据最有效的途径。它们把大部分处理数据的工作从实际的组件本身撤出来（请回想一下前面讨论的 MVC），并提供了一个公共数据对象类（例如 Vector 和 ArrayList）的包装器。

## 简单布局

就像在前面提到过的，布局替您处理组件在应用程序中的摆放。您的第一个问题可能是“为什么不能用像素告诉它应当在什么地方呢？”是的，您可以这样做，但是在窗口改变大小的时候，或者更糟一些情况，即用户改变其屏幕的分辨率的时候，亦或在有人想在其他操作系统上试用应用程序的时候，您立刻就会遇到麻烦。布局管理器把这些担心一扫而空。不是每个人都用相同的设置，所以布局管理器会创建“相对”布局，允许您指定组件相对于其他组件的摆放方式，决定事物改变尺寸的方式。这是好的部分：比听起来更容易。只要调用 `setLayout(yourLayout)` 设置布局管理器即可。后面对 `add()` 的调用可以将组件添加到容器中，并让布局管理器负责将它放在应当的位置上。

目前在 Swing 中包含了大量布局；看起来好象每次发布都会有一个新布局负责不同的目的。但是，有些经过实践检验的布局一直存在，而且会永远存在，我指的是永远 —— 因为从 1995 年 Java 语言的第一个发行版开始，就有这些布局。这些布局是：FlowLayout、GridLayout 和 BorderLayout。

**FlowLayout** 从左到右安排组件。当空间不足时，就移到下一行。它是使用起来最简单的布局，因此，也就是能力最弱的布局：

```
setLayout(new FlowLayout()); add(new JButton("Button1")); add(new  
JButton("Button2")); add(new JButton("Button3"));
```

### FlowLayout 实例

**GridLayout** 就像您想像的那样工作：它允许指定行和列的数量，然后在添加组件时把组件放在这些单元格中：

```
setLayout(new GridLayout(1,2)); add(new JButton("Button1")); add(new  
JButton("Button2")); add(new JButton("Button3"));
```

### GridLayout 实例

即使 **Swing** 中添加了许多新的布局管理器，**BorderLayout** 仍然是其中非常有用的一个。

即使有经验的 **UI** 开发人员也经常使用 **BorderLayout**。它使用东、南、西、北、中的概念在屏幕上放置组件：

```
setLayout(new BorderLayout()); add(new JButton("Button1"), "North"); add(new  
JButton("Button2"), "Center"); add(new JButton("Button3"), "West");
```

## BorderLayout 实例

## GridBagLayout

虽然上面的示例对于简单的布局来说很好，但是更高级的 UI 需要更高级的布局管理器。这是 GridBagLayout 发挥作用的地方。不幸的是，使用它的时候极易混淆、极为困难，每个曾经用过它的人都会同意这点。我也不能反对；但是除了它的困难之外，它可能是用 Swing 内置的布局管理器创建漂亮 UI 的最好方式。

以下是我的第一个小建议：在最新版的 Eclipse 中，有内置的可视化构建器，这个个小建议可以自动根据每个屏幕的需要来构建必需的 GridBagLayout 代码。请使用这个功能！它会节约无数为了让数字正确而浪费的时间。所以在我用这一节解释 GridBagLayout 如何工作、如何调整它才能让它做得最好时，建议您去找一个可视化构建器并生成代码。它会节约您的工作时间

## 事件

最后，我们来到 Swing 最重要的一部分：处理事件，对 UI 的交互作出反应。Swing 用事件/侦听器模型处理事件。这个模型的工作方式是：允许某个类登记到某个组件的某个事件上。登记到事件的这个类叫做侦听器，因为它等候组件的事件发生，而且在事件发生时采取行动。组件本身知道如何“激活”事件（即，知道它能生成的交互类型，以及如何让侦听器知道这个交互什么时候发生）。组件与包含有关交互信息的事件和类针对交互进行通信。

把技术方面的空谈放在一边，我们来看几个 **Swing** 中事件的实例。首先从最简单的示例开始，即一个 **JButton**，按下它的时候，会在控制台上输出“Hello”。

**JButton** 知道它什么时候被按下；这是在内部处理的，不需要代码处理它。但是，侦听器需要进行登记，以接收来自 **JButton** 的事件，这样您才能输出“Hello”。`listener` 类通过实现 `listener` 接口然后调用 **JButton** 上的 `addActionListener()` 做到这一点：

```
// Create the JButton
JButton b = new JButton("Button"); // Register as a listener

b.addActionListener(new HelloListener()); class HelloListener implements
ActionListener { // The interface method to receive button clicks
public void
actionPerformed(ActionEvent e) { System.out.println("Hello"); } }
```

**JList** 也用类似的方式工作。当有人在 **JList** 中选中什么时，您可能想把选中的对象输出到控制台上：

```
// myList is a JList populate with data
myList.addListSelectionListener(new
ListSelectionListener() { public void valueChanged(ListSelectionEvent e)
{ Object o = myList.getSelectedItemAt(); System.out.println(o.toString()); } } );
```

从这两个示例，您应当能够理解事件/侦听器模型在 **Swing** 中如何工作了。实际上，**Swing** 中的每个交互都是以这种方式处理的，所以通过理解这个模型，您就立即能够理解在 **Swing** 中如何处理每个事件，以及如何对用户可能抛给您的任何交互做反应了。

## 模型

现在，您应当了解了 Java 的集合（Collection），这是一组处理数据的 Java 类。这些类包括 ArrayList、HashMap 和 Set。大多数应用程序在反复处理数据时，经常用这些类。但是，当需要在 UI 中使用这些数据类时，出现了一个限制。UI 不知道如何显示它们。请先想一分钟。如果有一个 JList 和一个某种数据对象（例如 Person 对象）的 ArrayList，JList 怎样才能知道要显示什么？它是要显示某个人的名字，还是连名带姓一起显示？

这就是模型的概念发挥作用的地方了。虽然模型这个术语表达的范围更大，但是在本教程的示例中，我用 UI 模型这个术语描述组件用来显示数据的类。

在 Swing 中每个处理集合数据的的组件都采用模型的概念，而且这也是使用 and 操纵数据的首选方法。它清晰地把 UI 的工作与底层数据分开（请回想 MVC 示例）。模型工作的机制是向组件描述如何显示集合数据。我说的“描述”指的是什么呢？每个组件需要的描述略有不同：

- JComboBox 要求其模型告诉它把什么文本作为选项显示，以及有多少选项。
- JSpinner 要求其模型告诉它显示什么文本，前一个和下一个选择是什么。

- `JList` 也要求其模型告诉它把什么文本作为选项显示，存在多少选项。
- `JTable` 要求的更多：它要求模型告诉它存在多少列和多少行，列名称、每列的类以及  
在每个单元格中显示什么文本。
- `JTree` 要求它的模型告诉它整个树的根节点、父节点和子节点。

您可能会问：为什么要做这么些工作？为什么要把这些功能分开？请想像以下场景：您有一个复杂的 `JTable`，有许多列数据，您在许多不同的屏幕上使用这个表格。如果您突然决定删除某个列，那么怎么做会更容易呢？修改您使用的每个 `JTable` 实例中的代码？还是创建一个可以在每个 `JTable` 实例中使用的模型类，然后只修改这一个模型类呢？显然，所做的修改越少越好。

## 模型示例

我们来看看模型如何工作，在一个简单的 `JComboBox` 示例中使用模型。在 `JComboBox` 前面的演示中，我介绍了如何调用 `setItem()` 向数据中添加项目。虽然对于简单的演示，这样做可以接受，但是在实际的应用程序中很少这么用。毕竟，在有 25 个选项，而且选项不断变化的时候，您还真的想每次都调用 `addItem()` 25 次对这些选项进行迭代吗？当然不是。



JComboBox 包含一个方法调用 `setModel()`，它接受 `ComboBoxModel` 类的实例。应当用这个方法代替 `addItem()` 方法来创建 JComboBox 中的数据。

假设有一个 `ArrayList`，其中使用字母表作为其数据（“A”、“B”、“C”，等等）：

```
MyComboModel model = new MyComboModel(alphaList); myComboBox.setModel(model);

public class MyComboModel implements ComboBoxModel { private List data = new
ArrayList(); private int selected = 0; public MyComboModel(List list) { data =
list; } public void setSelectedItem(Object o) { selected = data.indexOf(o); }
public Object getSelectedItem() { return data.get(selected); } public int getSize()
{ return data.size(); } public Object getElementAt(int i) { return data.get(i); } }
```

采用模型时更好的地方是：您可以反复重用它。例如，假设 JComboBox 的数据需要从字母表变成 1 到 27 的数字。那么只用一行就可以实现这个变化：用新的数据 `List` 添加 JComboBox，不需要使用额外的代码：

```
myComboBox.setModel(new MyComboModel(numberList));
```

模型在 `Swing` 中是非常有好处的特性，因为它们提供了代码重用功能，而且使数据处理更加容易。更常见的应用是在大型应用程序中，服务器端开发人员创建和检索数据，并把数据传递给 UI 开发人员。如何处理这些数据和正确地显示它们，取决于 UI 开发人员，而模型就是实现这项任务的工具。

Swing 入门(一)(2009-04-21 00:02:20)

标签: [it](#)

分类: [Awt/Swing 学习](#)

(转自 <http://terrificwanjun.bokee.com/>)

```
package cn.edu.jnu.www;
```

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

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

```
import java.awt.*;
```

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

```
public class test {
```

```
    public static void main(String[] args) {
```

```
        JFrame a=new JFrame("中继数据库系统");
```

```
        Container c=new Container();
```

```
        //Swing 更强调容器的概念，一般不允许之间将组件放置到顶层容器中
```

```
        //而是放在容器框架中，而 awt 则是直接放的
```

```
        a.setSize(200,200);
```

```
        a.setLocation(100, 200);
```

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

```
        JButton b=new JButton("GO");
```

```
        c=a.getContentPane();
```

```
c.add(b, BorderLayout.SOUTH);

a.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

a.setVisible(true);

}

}
```

Swing 学习(二)(2009-04-21 00:03:22)

标签: [it](#)

分类: [Awt/Swing 学习](#)

(转自 <http://terrificwanjun.bokee.com>)

接触java 几天,自己试着做了第一个java 程序,其实只是一个简单的对话框窗口,不过做起还是费了半天功夫,主要是对一些语法还不太熟悉,幸亏有 CSDN 上的朋友相助,问题得以解决.

```
import java.awt.*;
```

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

```
public void class AboutDialog extends JDialog {           //这里误用 void,返回空值
```

```
    public AboutDialog() {
```

```

this.setTitle("About");           //窗体标题显示

    this.setSize(320, 200);       //窗体的大小


JLabel about = new JLabel("关于:JAVA 的一个窗口 :)");    //对话框内容

about.setHorizontalAlignment(SwingConstants.CENTER);    //内容显示在窗口的中
央

this.getContentPane().add(about, BorderLayout.CENTER);

}

public static void main(String[] args) throws HeadlessException {

    AboutDialog kk = new AboutDialog() ;

    kk.setVisible(true);           //原来的 show()显
示已过时

    kk.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

}

}

```

上面误用 void 生成的结果,有兴趣的朋友可以试试.另外查了一下 J2SE API,才知道 show( ) 方法已经被 setVisible()取代.

## 使用 Frame 创建窗体

在 Awt 中，Window 类没有边界和菜单栏，所以不能直接使用 Window 类来创建窗体，必须使用 Window 类的子类 Frame 来创建，创建代码为：

```
package cn.edu.jnu.www;

import java.awt.*;
import java.awt.event.*;

public class FrameTest {

    public static void main(String[] args) {

        // TODO 自动生成方法存根

        Frame myFrame=new Frame("Hello");

        //Frame 是带有标题和边界的顶层窗口
```

```
myFrame.setLocation(250, 150);
```

```
//myFrame.setLayout(new BorderLayout(10,20));
```

//设置窗体布局，BorderLayout 里面的两个参数用指定的组件之间的水平间距构造一个边界布局。

```
//BorderLayout 为 JAVA 中的默认窗体布局
```

```
//myFrame.setLayout(new FlowLayout(FlowLayout.LEFT));
```

//设置窗体布局为 FlowLayout，对齐方式居左对齐

```
//myFrame.setLayout(new GridLayout(3,2));
```

//设置窗体布局为 GridLayout，将窗体分为 6 块，3 行 2 列

```
myFrame.setSize(300,400);
```

```
Button myButton1=new Button("east");
```

```
Button myButton2=new Button("south");
```

```
Button myButton3=new Button("west");
```

```
Button myButton4=new Button("north");
```

```
Button myButton5=new Button("center");
```

```
//myFrame.add(myButton1, BorderLayout.EAST);
```

```
myFrame.add(myButton1,"East");
```

//两种方法都可以，但是要注意大小写

```
myFrame.add(myButton2,BorderLayout.SOUTH);
```

```
myFrame.add(myButton3,BorderLayout.WEST);
```

```
myFrame.add(myButton4,BorderLayout.NORTH);
```

```
myFrame.add(myButton5,BorderLayout.CENTER);
```

```
//myFrame.addWindowListener(new myWindowListener());
```

//使用适配器的方法实现监听器作用

```
//myFrame.addWindowListener(new yourWindowListener());
```

//使用匿名内部类的方法实现监听器的功能,只有含有适配器的事件才可以使用此方法

```
myFrame.addWindowListener(new WindowAdapter(){
```

```
    public void windowClosing(WindowEvent e){
```

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

```
    }
```

```
});
```

```
myFrame.setVisible(true);
```

```
//myFrame.show();//eclipse 不建议使用 show()方法
```

```
}
```

```
}
```

```
//使用 WindowAdapter 适配器类来实现监听器
```