AsWing 教程

iiley 著

Book version 1.0

目录

2.0 引言

这一章我们主要介绍如何使用 ActionScript3.0 构建用户界面。

当前，RIA 技术越来越成熟的情况下，构造界面的技术选择比以往多得更多了，在用 ActionScript1 写程序的时代，我们只能用 Flash IDE 制作界面。虽然用 Flash IDE 可以制作非常个性化的界面，但几乎每个按钮，都要用 IDE 来逐一绘制，这对程序员来说无疑是一种痛苦。于是，在 ActionScript2 时代，Macromedia[1] 在 Flash IDE 中为我们提供了一套 UI 组件——Version 2 Components（V2 组件），它包含了一些常用的组件，比如按钮、下拉列表、单选复选框等，方便了普通程序 UI 的构建。但是这套组件并不十分好用，于是 Flash 开源社区涌现出一批替代组件库：除了本章要向大家介绍的 AsWing 外，还有 EnFlash 和 ActionStep 等。凭借着良好的设计，易用的接口，加上 MTASC 编译器的推动，开源 UI 库在 Flash 开发社区得到了快速的发展。后来，到了 ActionScript3 时代，Flex 框架强势挺出，凭借其强大的体系架构、完备的组件库、紧密的开发环境支持，大部分开源 UI 库停止了 ActionScript3 版本的开发，仅有 AsWing 推出了 ActionScript3 版本，并且在架构上做了改进，自定义外观功能以及可视化布局器也相继推出，成为 Flex UI 框架的一个替代选择。

自 2005 年项目建立到现在，AsWing 经过了 3 年多的发展，历经了从 ActionScript2 到 ActionScript3 的迁移，国内外已有不少采用 AsWing 进行项目设计的成功案例，它们之中有个人实验作品，也有商业作品，有多人在线游戏，也有网络操作系统，不一而足。

那么作为 Flex UI 框架的替代选择，AsWing 与之不同的地方主要体现在哪几个方面呢？

1. AsWing 是纯 ActionScript3 代码实现，使用 AsWing 无需掌握 MXML 描述语言，不局限于 Flex 编译器，Flash 也可以编译。
2. AsWing 比 Flex 组件更灵活，由于 Flex 组件覆盖了 DisplayObjectContainer 的 addChild 等方法，使得普通显示元件和组件的配合极不方便，而 AsWing 组件可以和普通显示原件灵活搭配，可更方便地构造独特的外观。
3. AsWing 比 Flex 更轻量，生成的 swf 文件更小巧。
4. AsWing 组件更 MVC，大部分组件均采用 Model-View-Controller 模式设计，数据与界面更好的分离。
5. Flex 组件拥有更多的大型组件，目前 AsWing 缺少 AdvancedDataGrid 这样的复杂表格组件，当然 AsWing 也拥有 Flex 没有的一些组件，比

[1] 后来 Macromedia 被 Adobe 收购。

2. 案例展示可到 http://www.aswing.org/?page_id=7 查阅

如 ColorMixer,Accordian 等。

6. Flex 组件有更强大的可视化编辑器。虽然 AsWing 拥有 GuiBuilder 工具——可见即可得的编辑界面并生成 ActionScript3 代码，但是 Flex 的 Design 模式由于内嵌于 Flex Builder IDE 中，因此更方便使用。
7. Flex 组件主要通过 CSS 来自定义组件外观，AsWing 主要通过 SkinBuilderLAF 来做自定义，CSS 对于网页设计师更友好，SkinBuilderLAF 则更贴近美术设计人员（可直接用 Flash IDE 或者 Photoshop 来绘制外观元素）。
8. Flex 组件与 Flex 框架紧密结合，提供了整套 RIA 技术方案，AsWing 保持单纯 UI 框架结构，不耦合于任何一种应用程序框架。

由以上几点读者可了解到 AsWing 与 Flex UI 框架的主要区别，由于 Flex 的书籍已大量充斥于市面，加上本书作者均更熟悉 AsWing，因此本书将完全采用 AsWing 来讲解界面构建，为Flash 应用开发提供 Flex 之外的另一种解决方案。

笔者认为，已经成熟的 2 个框架之间，没有绝对的孰优孰劣，适合于自己项目，适合于自己的开发习惯，熟练掌握了它的使用方法，于你而言它就是好的框架。

需要注明的是，截止本书初稿完成之日，AsWing 的最新版本为 1.5。你读到此书的时候，2.0 应该已经趋于发布。

2.1 小试身手，hello 日程

按照惯例，第一节快速编写并运行一个程序。

笔者这里就不 Hello World 了，都被无数人叫腻了，这里对我们后面将要讲解的例子打个招呼。Hello 日程，对，在这一章，我们会通过一个日程管理程序的编写来讲解界面构建。

为什么要用日程管理程序来做例子呢，听起来好像很没趣的一个程序，但是我想了很久都没想到更好的例子。讲应用程序界面，总不能用一个 3D 游戏引擎来做例子吧，再说我也不懂 3D 引擎，由于日程管理程序大部分工作量在于构建界面，所以以此为例子可以更集中于主题。并且，日程管理程序对大多数人的日常生活也许会有那么点用处，我会尽量把这个例子程序设计得有趣一些，并且易用一些。

切入正题吧，用 FlexBuilder 新建 ActionScript 项目，项目名为 HelloScheme，注意在 Library Path 中不要忘记了添加 AsWing 项目或者 AsWing.swc。

编写主类代码如下：

HelloScheme.as

```
package {

import flash.display.Sprite;

import org.aswing.AsWingManager;
import org.aswing.JFrame;
import org.aswing.JLabel;

public class HelloScheme extends Sprite{

    public function HelloScheme(){
        //初始化AsWing
        AsWingManager.initAsStandard(this);
        //创建一个JFrame（窗体）
        var frame:JFrame = new JFrame(this, "Hello 日程");
        //创建一个JLabel（标签）
        var label:JLabel = new JLabel("Hello 日程");
        //把标签加入到窗体中
        frame.getContentPane().append(label);
        //设置窗体大小为300*200
```

```
            frame.setSizeWH(300, 200);
            //显示窗体
            frame.show();
        }
}
}
```

程序中各行的意义已经通过注释讲明，这里就不多说了，具体的细节留到后面章节再逐一讲解。编译运行此程序后会得到如下画面（图 1）。

（图 1）

一个完整的 AsWing 程序就完成了，虽然几乎没有任何功能，但是还是比普通的语言介绍的 Hello World 复杂了一些，足足有 20 来行代码。其中涉及到窗体的创建，标签的创建，向容器添加子组件，设置窗体大小等内容。可以看到，一个 AsWing 程序，和普通 ActionScript3 程序没有什么两样，就是 import 几个 AsWing 的类并使用他们而已，完全是纯代码的事情。这里你不用急着理解其中每一行代码的含义，后面章节我们会慢慢讲解相关内容。

2.2 创建主界面 (主界面的规划)

在开始编码之前，免不了要先计划一下。何为日程管理软件？我们首先要确定的是：它需要提供哪些功能，大体的界面布局如何？

对于功能设计，不是本章的要点，这里我们就不加详述，直接列出经过仔细考虑后的功能点：

1. 每个日程，需包含日程名称，日程说明，开始日期和时间，预计进行时间长度，重要程度，日程状态（计划，进行中，取消，已完成，已耽误）。
2. 可以添加，删除，编辑日程
3. 可以对日程依据各种属性进行排序
4. 对于紧急或重要的日程进行区分（颜色区分）。

可以看到，功能并不复杂，也就是一个简单的小程序，它的重要工作量不在逻辑而在 UI，所以接下来我们需要认真策划一下 UI 的设计。按照大多应用程序的操作习惯，我们可以把所有的控制和编辑功能菜单安排在顶端，再增加一个工具栏，提供一些常用的工具按钮比如添加，删除，编辑等按钮，中间部分放置日程列表，如果需要，最下端还可以放一个状态栏，目前我们还不能肯定需不需要状态栏，我们可以先预留出位置，方便以后添加。如此设定，我们可以大致得出一个 UI 结构图如下。

（图 2）

可见此布局把界面分成 4 个部分，顶端是菜单和工具栏，中间为日程列表，下面是状态

栏。我们可以为每个部分单独创建一个容器，然后对这各个容器布局，布局定好了之后只需为各个单独容器添加内容即可。

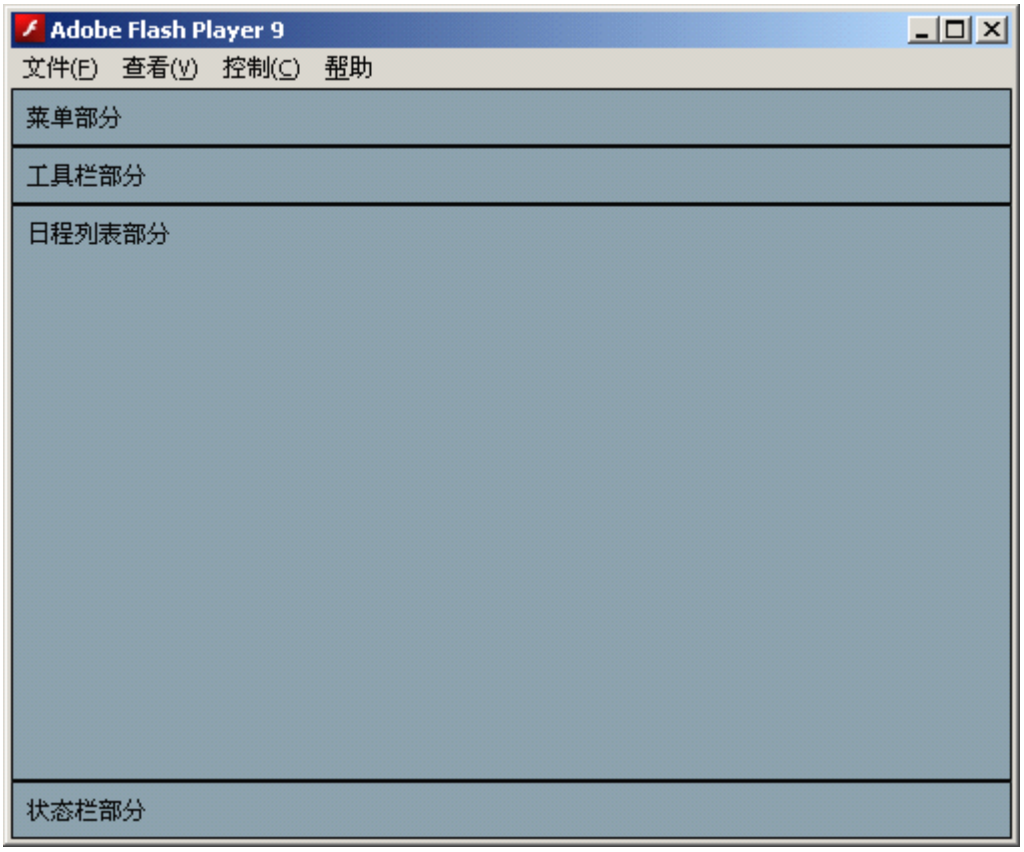
终于可以开始编码了，创建项目之前，我们得为这个项目取个名字，就简单地叫 SchemeManager 好了（名字，称谓而已嘛）。在 Flex Builder 中创建 ActionScript 项目 SchemeManager，注意 Library Path 中不要忘记了添加 AsWing 项目或者 AsWing.swc，Main Application file 我们定为 SchemeManager.as，编写代码如下：

SchemeManager.as

```
package {

import flash.display.Sprite;

import org.aswing.AsWingManager;
import org.aswing.BorderLayout;
import org.aswing.Container;
import org.aswing.JLabel;
import org.aswing.JPanel;
import org.aswing.JWindow;
import org.aswing.SoftBoxLayout;
import org.aswing.border.LineBorder;

public class SchemeManager extends Sprite{

    //菜单容器
    private var menuContainer:JPanel;
    //工具栏容器
    private var toolContainer:JPanel;
    //日程列表容器
    private var tasksContainer:JPanel;
    //状态栏容器
    private var statusContainer:JPanel;

    private var mainWindow:JWindow;

    public function SchemeManager(){

        AsWingManager.initAsStandard(this);

        menuContainer = new JPanel();
        toolContainer = new JPanel();
        tasksContainer = new JPanel();
        statusContainer = new JPanel();
```

```
        mainWindow = new JWindow();

        //得到窗口容器
        var pane:Container = mainWindow.getContentPane();
        pane.setLayout(new BorderLayout());

        //顶端部分
        var top:JPanel = new JPanel(new SoftBoxLayout(SoftBoxLayout.Y_AXIS));
        top.append(menuContainer);
        top.append(toolContainer);
        pane.append(top, BorderLayout.NORTH);
        //中间部分
        pane.append(tasksContainer, BorderLayout.CENTER);
        //低端部分
        pane.append(statusContainer, BorderLayout.SOUTH);

        //给每个部分添加标签以观察布局
        addLabel(menuContainer, "菜单部分");
        addLabel(toolContainer, "工具栏部分");
        addLabel(tasksContainer, "日程列表部分");
        addLabel(statusContainer, "状态栏部分");

        //设置窗口大小为舞台大小
        mainWindow.setSizeWH(stage.stageWidth, stage.stageHeight);
        //显示窗口
        mainWindow.show();
    }

    private function addLabel(pane:JPanel, label:String):void{
        pane.append(new JLabel(label));
        //给容器设置一个边框以观察大小范围
        pane.setBorder(new LineBorder());
    }
}
}
```

代码中 mainWindow 是整个界面的根容器，根容器一般用 JRootPane 及其子类（它们是 JPopup, JWindow, JFrame 等）担当。为了方便查看布局是否正确，我给目前还空着的每个容器都添加了标签和边框。代码中关于 SoftBoxLayout 和 BorderLayout 的含义和用法会在后面章节介绍，这里可以暂时不去理解它。运行这个程序，我们可以得到如下结果：



（图 3）

你也许还不能完全理解这段代码的所有部分，没关系，你大可以直接往后面阅读，具体的知识点本章会依次介绍。

本节知识点：

显示原件，组件，容器，布局管理器：AsWing 界面中，最基本的元素是组件，对应的类是 Component，容纳 Component 的是容器 Container，Container 是 Component 的直接子类。即是说，Container 也是组件，只不过它比组件多出 append,remove,setLayout 等这几个与容纳和布局相关的方法，布局管理器 LayoutManager 为容器内的组件做布局工作，它负责管理和配置容器中组件的位置与大小。由于 AsWing AS3 版本在设计之初，作者尝试使用过 Flex 组件，对 Flex 组件覆盖 DisplayObjectContainer 的 addChild 方法并且只允许 IUIComponent 被加入容器的做法嗤之以鼻，因此 AsWing 组件对于普通显示原件 DisplayObject 及其子类的友好度非常高。在一些情况下，开发者想要给组件里添加一个图片，或者一个动画，可以直接用 Component.addChild 方法，这是可以正常运作的。

JPanel：JPanel 是 AsWing 中最常用的容器组件，它是 Container 的直接子类，通常作为透明或非透明容器用（setOpaque 方法可以设置它的透明属性）。

JWindow：JWindow 是 AsWing 中常用的窗口组件，它是 JPopup 的子类，可以做弹出窗口用，对于单窗口程序，它也可以作为根组件使用。

JLabel：JLabel 是标签组件，它用于显示一个字符串。

LineBorder：线框，用于在组件周围绘制一个矩形边框。

方法：**append 方法**用于把一个组件加入到一个容器中，**setBorder 方法**可以为一个组件设置边框，**setSizeWH 方法**设置组件的高和宽，**setLayout 方法**设置容器的布局管理器。

2.2.1 工具栏的构建 （JToolBar，JButton，事件）

上一节完成的界面，只是一个大体布局，离我们想要的最终效果还差很远。一个完整的应用程序界面是不可能一蹴而就的，我们可以按模块来分别进行开发。所谓万事开头难，我的习惯是选择从较为简单的部分——工具栏入手。

工具栏通常会放置最常用的工具按钮，按我们现在的规划，工具栏应该至少包含添加日程，删除日程，编辑日程。目前能确定的就这些，以后如果还需要更多，我们可以再添加。AsWing 有提供工具栏容器组件 JToolBar，按钮组件 JButton。目前的工具栏，我们只需要把 3 个 JButton 横排列在 JTooBar 之中即可。

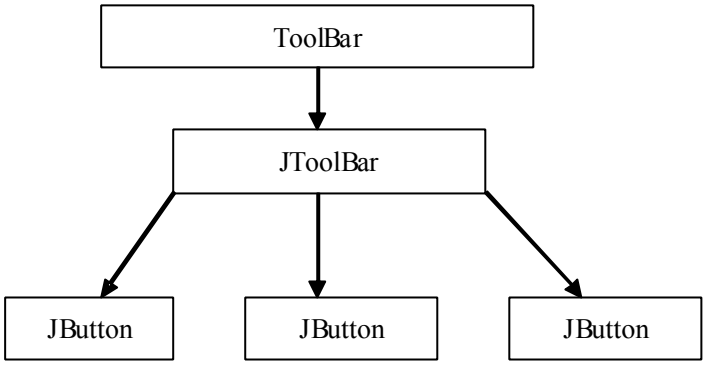
为了尽量模块化使得后面容易修改，我们把工具栏编写成一个类，如下：

ToolBar.as

```
package book.scheme{

import org.aswing.BorderLayout;
import org.aswing.JButton;
import org.aswing.JPanel;
import org.aswing.JToolBar;

public class ToolBar extends JPanel{

    private var bar:JToolBar;
    private var addButton:JButton;
    private var removeButton:JButton;
    private var editButton:JButton;

    public function ToolBar(){
        super();
        setLayout(new BorderLayout());

        bar = new JToolBar(JToolBar.HORIZONTAL, 2);
```

```
        append(bar, BorderLayout.CENTER);

        addButton = new JButton("添加日程");
        removeButton = new JButton("删除日程");
        editButton = new JButton("编辑日程");

        bar.appendAll(addButton, removeButton, editButton);
    }
}
}
```

我们把这个类命名为 book.scheme.ToolBar，由于工具栏部分是需要包含一些**列按钮**，或者更多其他东西（以后可能会添加），因此我们让它继承自容器类 JPanel，这样能方便地给它添加任何组件元素。目前的实现，我们给它添加了一个 JToolBar，并给 JToolBar 添加了 3 个按钮。它们之间的关系犹如下图：



（图 4）

ToolBar 容器包含了一个 JToolBar，JToolBar 包含了 3 个按钮。

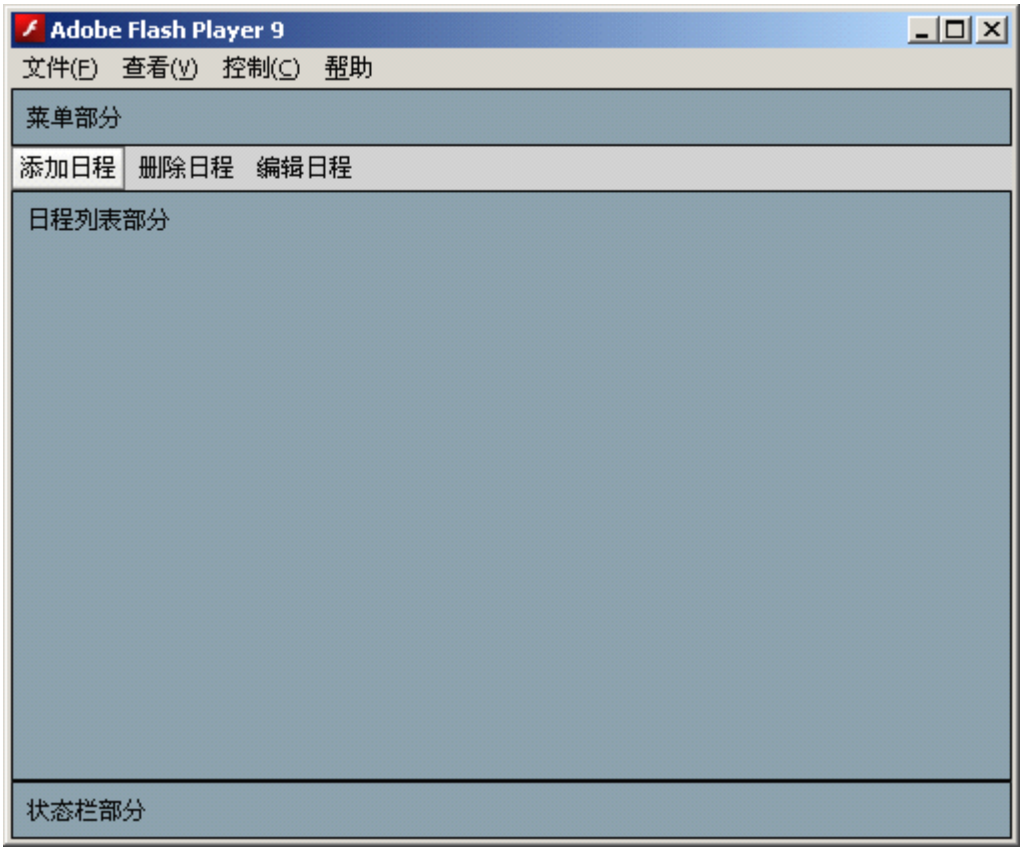
JToolBar 的构造函数接受两个参数，形式为 (orientation:int, gap:int)，第一个参数 orientation 代表排列方式，它可以是 JToolBar.HORIZONTAL 指明横向排列或者 JToolBar.VERTICAL 指明竖向排列，gap 是指 JToolBar 内部的组件之间的间隔（像素单位，注：AsWing 中，距离，位置，大小等度量都以像素为单位）。上面代码中，我们指明创建一个横向排列，间隔为 2 个像素的工具条。之后创建 3 个按钮，然后通过 appendAll 把它们一次性添加到工具条中。appendAll 的作用与 append 相同，都是用于把一个组件添加到一个容器里，只是 appendAll 接受可变参数数量，可以一次添加任意个组件，本来需要调用 3 次 append，采用 appendAll 只需要调用一次。

JButton 构造函数接受两个参数，形式为 (text:String, icon:Icon)，第一个参数指按钮要显示的文本，第二个参数指要显示的图标。

此段代码中关于布局管理部分的原理，比如为何 ToolBar 要使用 BorderLayout，读者当

前可以不予理会，在 2.3 节我们会集中讲解组件的布局。

为了快速查看这个类的效果，我们可以修改 SchemeManager 的语句 *top.append(toolContainer)*;为 top.append(new ToolBar());，编译并运行后，可以得到如下画面：



（图 5）

当鼠标移动到 3 个按钮上时，可以看到按钮会弹起，点击会使它凹陷，就像大多应用程序的工具栏按钮那样。只是目前点击这些按钮，除了按钮外观有变化外，没有任何其他反应，我们也不知道如何处理用户的点击事件。为了当用户按下按钮时，我们能做相应的工作，比如点击“添加日程”我们就给添加一个日程，由此，我们需要监听按钮的事件来达到这个目的。

AsWing 的 JButton 是继承自 AbstractButton 而来，AbstractButton 有三个事件，分别为：

1. AWEvent.ACT
2. InteractiveEvent.STATE_CHANGED
3. InteractiveEvent.SELECTION_CHANGED

其中 AWEvent.ACT 是指按钮采取行动事件，也就是当用户单击按钮时触发的事件；InteractiveEvent.STATE_CHANGED 是当按钮状态改变时触发的事件；InteractiveEvent.SELECTION_CHANGED 是当按钮选中状态改变时触发的事件。我们需要处理的是单击事件，因此我们可以监听 AWEvent.ACT，我们目前还不涉及具体的日程处理工作，因此我们可以在事件监听中简单的向控制面板输出一些信息来观察事件的执行。

给 ToolBar 添加如下代码：

```
        addButton.addEventListener(AWEvent.ACT, __addTask);
        removeButton.addEventListener(AWEvent.ACT, __removeTask);
```

```
        editButton.addEventListener(AWEvent.ACT, __editTask);
    }

    private function __addTask(e:AWEvent):void{
        trace("添加一个日程");
    }

    private function __removeTask(e:AWEvent):void{
        trace("删除一个日程");
    }

    private function __editTask(e:AWEvent):void{
        trace("编辑日程");
    }
```

可以看到，AsWing 的事件监听和处理方式是和 AS3 本身的事件机制一样的，这是因为组件类 Component 继承自 Sprite，也就是 EventDispatcher 的子类，因此他拥有 EventDispatcher 的机制。

编译并调试修改后的程序，当你点击这三个按钮时，会在控制台看到各自相应的输出。由此，我们的程序开始和用户互动了。

本节知识点：

JToolBar：JToolBar 是一个工具栏容器，它可以横向或者竖向排列加入其中的组件，并会使得加入其中的按钮具有一般应用程序工具栏按钮的效果，即鼠标滑过时弹起，滑开时隐藏按钮边框，鼠标在按钮之上时按钮保持通常的行为。

JButton：JButton 是普通按钮组件，它继承自 AbstractButton，拥有 AbstractButton 的所有行为和方法以及事件。除了 JButton，还有开关按钮 JToggleButton，单选按钮 JRadioButton，复选框 JCheckBox 等都是 AbstractButton 的子类。

AsWing 事件：AsWing 组件的事件和 AS3 本身的事件机制是一样的，同样是采用 EventDispatcher 的机制。也是由于这个原因，所以 AsWing 的组件事件都是继承自 Event 类。此章假定读者已经掌握了 AS3 编程基础，所以关于事件处理这里就不详细叙述，每个特定事件的特点会在例子程序涉及到时进行单独讲解，没有涉及到的，可以阅读 api 文档进行了解。为了方便使用，AsWing 对一些常用的事件监听进行了封装，比如 AWEvent.ACT 事件，AsWing 在 AbstractButton 类中有如下封装：

```
    public function addActionListener(listener:Function,
priority:int=0, useWeakReference:Boolean=false):void{
    addEventListener(AWEvent.ACT, listener, false, priority,
useWeakReference);
    }
```

因此上面的程序中，我们可以把 *addButton.addEventListener(AWEvent.ACT,*写成 *addButton.addActionListener(*，二者效果是一样的。

AWEvent.ACT 事件：此事件是 AsWing 组件常用的事件之一，它被定义为执行动作之一，由此可见它适用于绝大多数交互组件，大部分组件的默认行为，都会触发此事件，比如按钮单击，菜单选择，文本框输入确定等。具体那些组件会触发此事件，可以查看 api 文档或者简单检查组件类是否含有 addActionListener 方法即可得知。

InteractiveEvent.STATE_CHANGED：此事件也是应用非常广泛的事件之一，它的含义为状态改变。不同的组件，对状态有不同的定义。对按钮而言，状态属性有

1. 是否可用 enabled,
2. 鼠标是否滑上 rollOver,
3. 是否被按下 pressed,
4. 是否被释放 released,
5. 是否被选中 selected

五个状态属性，任何一个状态属性发生变化时，都会触发这个事件。关于按钮状态模型，可以阅读 ButtonModel 接口的 api 文档进行更全面的了解。

对于其他拥有此事件的组件，状态属性可能不一样，具体请参考 api 文档。

InteractiveEvent.SELECTION_CHANGED 事件：对于拥有选择状态的组件，都会拥有此事件，可以看出，上面状态改变事件实际上包含了此事件。由于选择状态的改变是非常重要的事件，常常需要单独监听这一变化，因此它被单独提出来形成一个事件。

2.2.2 菜单的构建（JMenuBar, JMenu，JMenuItem)

这一节我们来完成菜单的构建。目前而言，我们还不完全确定需要哪些菜单，因此我们先创建能想到的那部分。主菜单分为“编辑”和“帮助”。编辑包含“新建日程”，“编辑日程”，“删除日程”；“帮助”现在只包含一个“关于”。

和工具栏的做法一样，我们也把菜单部分写成一个独立的类。代码如下：

Menu. as

```
package book.scheme{

import org.aswing.BorderLayout;
import org.aswing.JMenu;
import org.aswing.JMenuBar;
import org.aswing.JMenuItem;
import org.aswing.JPanel;
import org.aswing.event.AWEvent;
```

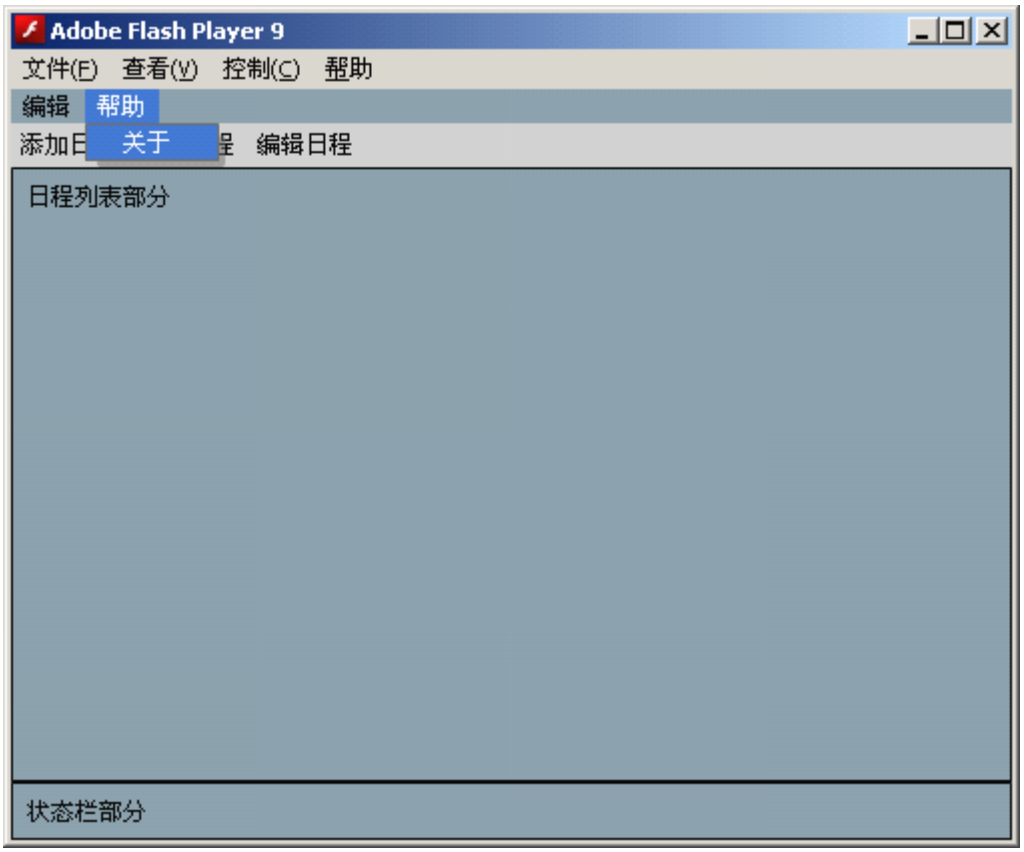
```
public class Menu extends JPanel{

    private var bar:JMenuBar;
    private var editMenu:JMenu;
    private var addTaskMenu:JMenuItem;
    private var editTaskMenu:JMenuItem;
    private var deleteTaskMenu:JMenuItem;
    private var helpMenu:JMenu;
    private var aboutMenu:JMenuItem;

    public function Menu(){
        super();
        setLayout(new BorderLayout());

        bar = new JMenuBar();
        editMenu = new JMenu("编辑");
        addTaskMenu = new JMenuItem("添加日程");
        editTaskMenu = new JMenuItem("编辑日程");
        deleteTaskMenu = new JMenuItem("删除日程");
        helpMenu = new JMenu("帮助");
        aboutMenu = new JMenuItem("关于");

        append(bar, BorderLayout.CENTER);

        bar.append(editMenu);
        bar.append(helpMenu);

        editMenu.append(addTaskMenu);
        editMenu.append(editTaskMenu);
        editMenu.append(deleteTaskMenu);

        helpMenu.append(aboutMenu);

        addTaskMenu.addActionListener(__menuAction);
        editTaskMenu.addActionListener(__menuAction);
        deleteTaskMenu.addActionListener(__menuAction);
        aboutMenu.addActionListener(__menuAction);
    }

    private function __menuAction(e:AWEvent):void{
        var menu:JMenuItem = e.currentTarget as JMenuItem;
        trace(menu.getText() + " clicked!");
    }
}
```

```
}
```

可以看出，创建菜单的方式和工具栏大同小异，都是使用了一些基本的创建实例，然后 append 到容器的做法。不同的地方是，JMenu 需要放置到 JMenuBar 容器中，而 JMenuItem 需要放置到 JMenu 之中。细心的读者可能已发现，在添加三个日程菜单的时候，这里没有使用 appendAll 方法而是调用了三次 append 方法，这是因为目前的版本（AsWing 1.4）的 JMenu 没有提供 appendAll 方法，是容器的话继承自 Container 都会拥有 appendAll 方法，但是 JMenu 并不是 Container 的子类，因为在内部实现上 JMenu 并不直接包含它的子项，而是通过一个 JPopupMenu 来代理，行为上也是如此——点击 JMenu 时弹出一个包含其子项的面板，所以 JMenu 没有拥有 Container 的方法是正常的。JMenuBar 是专门用来放置 JMenu 的容器，它是正规继承自 Container 的容器。

为了立即看到菜单反应，这里我们也给所有菜单项添加了事件监听，不同的是，我们没有为每个菜单项单独编写监听函数，而是统一监听到一个函数。在监听函数里，通过 *Event.currentTarget* 来获取被触发的组件，然后输出对应的字符提示。

JMenuItem 是 AbstractButton 的子类，可以看到它的创建方式和 JButton 类似。为了看到菜单的运行效果，我们改写 SchemeManager 的语句 *top.append(menuContainer)*; 为 top.append(new Menu());，编译并运行后，可以得到如下画面：



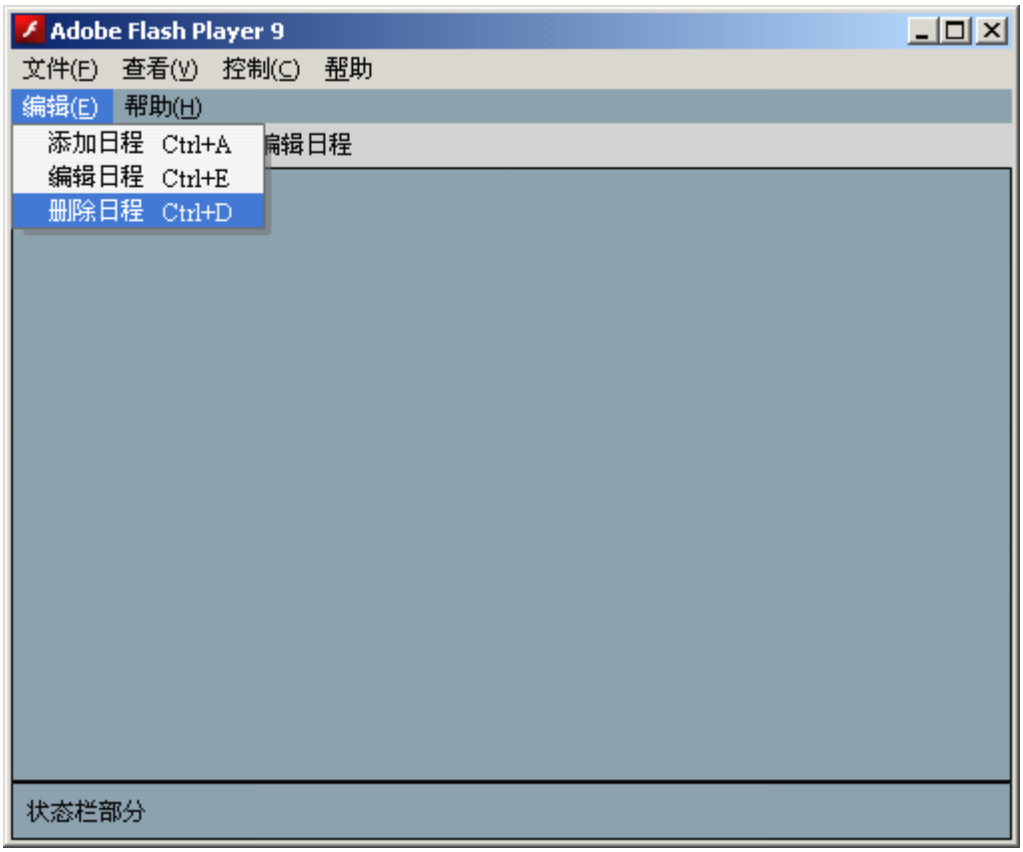
（图 6）

现在操作按钮项会得到相应的输出。但是这菜单似乎少了些什么，对了，没有助记键和快捷键的支持。通常，应用程序的菜单都会有助记键支持，比如 Alt+F 打开文件菜单，Alt+E

打开编辑菜单，快捷键 Ctrl+O 直接对应显示出来的“打开”菜单项。AsWing 也提供了这样的功能，给 Menu 类改变 JMenu 的创建代码，并添加菜单项的 accelaration 设置代码，如下：

```
        editMenu  = new JMenu("编辑(&E)");
        helpMenu  = new JMenu("帮助(&H)");
        aboutMenu  = new JMenuItem("关于(&A)");

        addTaskMenu.setAccelerator(new KeySequence(
               KeyStroke.VK_CONTROL, KeyStroke.VK_A));
        editTaskMenu.setAccelerator(new KeySequence(
            KeyStroke.VK_CONTROL, KeyStroke.VK_E));
        deleteTaskMenu.setAccelerator(new KeySequence(
            KeyStroke.VK_CONTROL, KeyStroke.VK_D));
        fscommand("trapallkeys", "true");
```

再运行程序，得到如下界面：



（图 7）

可以看到，和通常的应用程序菜单一样了。Ctrl+A 可以直接触发“*添加日程 clicked!*”的输出（注意你必须先点击到一个组件让窗口获得键盘焦点）。但是 Alt+E 却无法打开编辑菜单，这是因为 Flash Player 到目前为止还没有提供 Alt 键的支持，所以 AsWing 默认用 Ctrl+Shift 来代替 Alt，所以如果你按 Ctrl+Shift+E，就会打开编辑菜单，当然，这也是可以设置的，调用 *KeyboardManager.setDefaultMnemonicModifier(keyCodes:Array)*可以改变这一设置。为了能监听到 Ctrl 键的事件，最后我们还调用了 *fscommand("trapallkeys",*

"true")，这并不是必须的，这个语句的作用是会让 flash player 能把所有 Ctrl+其他按键的信息接收进来，包括浏览器或者播放器本身菜单的快捷键也会被触发进来，也就是说，如果浏览器或者播放器菜单已经使用了 Ctrl+X 快捷键，如果你没有调用这个语句，则 Ctrl+X 事件会被浏览器或者播放器完全吃掉，flash 程序就监听不到了，反之，则能监听到。

助记键的设置方法是通过创建菜单时在助记符前面加“&”符号实现的。快捷键通过 setAccelerator(acc:KeyType):void 方法实现。上面的代码中，我们使用了 KeySequence 来实现连续按键的快捷键，它的构造函数接受多个单按键 KeyStroke 连续排列组成。

本节知识点：

JMenuBar：菜单条唯一的作用是容纳菜单，菜单作为 Component 的子类，理论上可以放置于任何一个 Container 及其子类中，但是菜单条不仅仅提供对菜单的包含作用，它还提供了菜单的键盘导航作用——选中任何一个菜单时按左右键可以在同级别其他菜单中切换。

JMenu：菜单是菜单项（JMenuItem）的子类，因此菜单也拥有菜单项的所有功能，如果把 JMenuItem 比作 Component，JMenu 就好比 Container。菜单还可以包含菜单（子菜单）。然后菜单并不是一个正规的 Container，他对菜单项的容纳是通过一个 JPopupMenu 来实现的，因为菜单非选择状态下并不需要显示出它的子项，只有在选中时，才弹出一个包含了其子项的 JPopupMenu。

JMenuItem：菜单项是 AbstractButton 的子类，所以菜单项的行为和按钮行为一样，所拥有的方法也大体相同。当然，AsWing 也提供了单选菜单 JRadioButtonMenuItem 和复选菜单 JCheckBoxMenuItem，它们的行为与 JRadioButton 和 JCheckBox 一样。

助记键和快捷键：由于以往 Flash 程序大多是嵌入网页在浏览器运行，而浏览器占用了绝大多数的助记键和快捷键，给 Flash 程序提供助记键和快捷键的意义不大，反而会混淆用户的感知。所以绝大多数 Flash UI 库都没有提供完整助记键和快捷键支持，包括最新的 Flex3 组件库也是如此。而 AsWing 是笔者所见过的对此支持最完整的 UI 库，这是因为随着 AIR 应用的逐渐发展，用 ActionScript 开发独立桌面应用程序的情况越来越多了，完整的助记键和快捷键支持将有助于 ActionScript 应用程序的界面操作友好度。

2.2.3 日程表格的创建（JTable，JScrollPane)

主界面的 4 个部分，我们已经完成工具栏和菜单两个部分了。这一节我们来完成最重要的一部分——日程表格部分。由于这部分几乎是整个程序最复杂的部分，所以笔者不会尝试在一个小节的内容中就完成它。这一节我们只完成一个大体框架。

日程若是多了仅仅一屏肯定显示不完，所以我们得有滚动条支持，按照惯例，我们创建类 TaskTable 代码如下：

TaskTable.as

```
package book.scheme{

import org.aswing.BorderLayout;
import org.aswing.JPanel;
import org.aswing.JScrollPane;
import org.aswing.JTable;

public class TaskTable extends JPanel{

    private var table:JTable;
    private var scrollPane:JScrollPane;

    public function TaskTable(){
        super();
        setLayout(new BorderLayout());

        table = new JTable();
        scrollPane = new JScrollPane(table);
        append(scrollPane, BorderLayout.CENTER);
    }
}
}
```

TaskTable 照例继承自 JPanel，里面装一个 JScrollPane，这个 JScrollPane 包含着一个 JTalbe——日程用的表格。就这么几行，目前也还看不到效果，因为表格里没有数据。这个类虽然简单，但是涉及到了两个比较复杂的组件，JTable 和 JScrollPane，因此用一节专门讲讲它们，会有助于对后续内容的理解。

创建 JTable 实例比较简单，JTable 构造函数接受一个 TableModel 参数，可省略。我们现在还没有 Model，暂时省略，以后可以通过 setModel 方法来设置。JScrollPane 的构造函数形式为 *JScrollPane(viewOrViewport:*, vsbPolicy:int, hsbPolicy:int)*，第一个参数没有限制类型，它可以是一个 Component 或者 Viewportable 实例，后面两个参数指定滚动条的出现判定，是总不出现，还是总是出现亦或是根据需要出现，默认是根据需要出现，各参数含义具体设定请参见 api 文档。

本节知识点：

JTable：表格组件，把数据 Model 用行列的方式展现，每个列（Column）有个表头（Header）。相关类有 TableModel——表格模型，表格数据的提供者；JTableHeader——表头，TableCell

——表格单元格（单元格数据显示器）等等。

JScrollPane：滚动面板 JScrollPane 是 AsWing 中最重要的面板组件之一，它提供了整套用于可滚动组件的方案，使用简便，可定制程度高。JScrollPane 和一个 Viewportable 协同工作，提供显示内容的滚动能力。通常，大型的组件自身都实现了 Viewportable 接口，因此直接在 JScrollPane 构造函数的第一个参数传递这些组件即可完好工作。对于没有实现 Viewportable 的组件，用户可以自己实现，或者使用 JViewportable 来包装这个组件，以提供一个基于像素的滚动能力。JScrollPane 本质上只是一个含有水平和垂直滚动条的容器，它会包含一个 Viewportable，滚动内容包含在 Viewportable 容器中，具体滚动的幅度和粒度，是通过 Viewportable 来获得，所以通常，重点在于 Viewportable 提供的关于滚动的数值。AsWing 已有的组件中，JTextArea， JList， JTree 和 JTable 以及 GridList 都实现了 Viewportable 接口，它们自己管理和提供滚动信息。JViewport 则是一个默认的视口实现，用于把一个普通大组件根据它的 preferredSize（后面章节会讲解 preferredSize 的概念）来基于像素级别滚动。在 2.7.1 节，我们还会详细介绍 JScrollPane。

2.2 节完成的源代码可以在光盘目录下找到，文件名为 SchemeManager_2.2.zip，本章将一直以大节为单位存放项目源代码，以便于查阅和实践。

2.3 新建日程

在上面一节，我们把界面的架子搭了起来，现在要开始着手细节。从无到有，自然要先编写新建日程的功能。这一节，我们要动真格的了。

这里，新建日程的界面，应该是什么样的呢，读者可以自行想想。……，……，在确定日程数据的结构之前，不可能想象出确切的界面。因此，对于这一部分，我们可以按照这个顺序来进行设计，首先确定数据结构（Model），其次确定显示界面（View），最后确定界面和数据的关联——控制器（Controller）。

在 2.2 节，我们已经分析过，日程应该包含这些属性：日程名称，日程说明，开始日期和时间，预计进行时间长度，重要程度，日程状态（计划，进行中，取消，已完成，已耽误）。在进行具体的编码之前，笔者对日程状态又有了新的考虑，对于状态，特别是“已完成”和“已耽误”两个状态，本意应该是前者表示成功完成，后者表示没有进行，对于已经开始进行但未能按期完成的情况，这两种状态都难以准确表达。因此，这里我把日程状态修正为**计划**，**取消**，**进行中**，**中断**，**完成**，**耽误** 6 种状态，并增加状态注释属性，对于复杂情况，可以通过注释来详细说明情况。从字面意思可以看出，计划和取消，属于日程进行之前的状态，进行中指日程正在进行，中断，完成和耽误属于日程时间已经过去的状态。

由此，我们可以编写日程及相关类代码如下：

Task.as

```
package book.scheme{
/**
 * 日程数据
 */
public class Task{
    //名称
    public var name:String;
    //描述
    public var description:String;
    //开始日期事件
    public var startTime:Date;
    //进行时间长度，单位小时
    public var processTime:Number;
    //重要性[0, 10] 0最低，10最重
    public var importance:int;
    //状态
    public var status:String;
    //状态备注
    public var statusComment:String;
}
```

```
}
```

Task.as

```
package book.scheme{

/**
 * 日程的计划，取消，进行中，中断，完成，耽误6种状态
 */
public class TaskStatus{
    //计划
    public static const PLAN:String = "计划";
    //取消
    public static const CANCEL:String = "取消";
    //进行中
    public static const PROCESSING:String = "进行中";
    //中断
    public static const INTERRUPTED:String = "中断";
    //完成
    public static const FINISHED:String = "完成";
    //耽误
    public static const MISS:String = "耽误";
}
}
```

数据结构不是本书的重点，因此这里只简要介绍一下这两个类的设计思路。Task 类含有的各属性均设计为 public，方便读写。importance 属性设计为整型，方便以后排序。status 设计为字符串类型，方便显示。TaskStatus 类为 status 可能的各种值定义常量。

2.3.1 新建日程的界面（布局原理，JTextComponets, JComboBox）

上一节我们完成了数据结构（Model），这一节我们来完成显示界面（View），也就是设置日程属性的表格窗口——新建日程的界面。

按照一般应用程序的习惯，我们把此界面设计成一个可弹出的窗口界面，也就是说当用户点击“新建日程”的时候，我们需要弹出一个窗口界面让用户设置新日程的所有属性，当用户设置完成点击“确定”按钮时，程序则通过用户所填入的数据生成一个日程对象（Task Intance）。按此要求，我们编写如下类代码：

CreateTaskPane.as

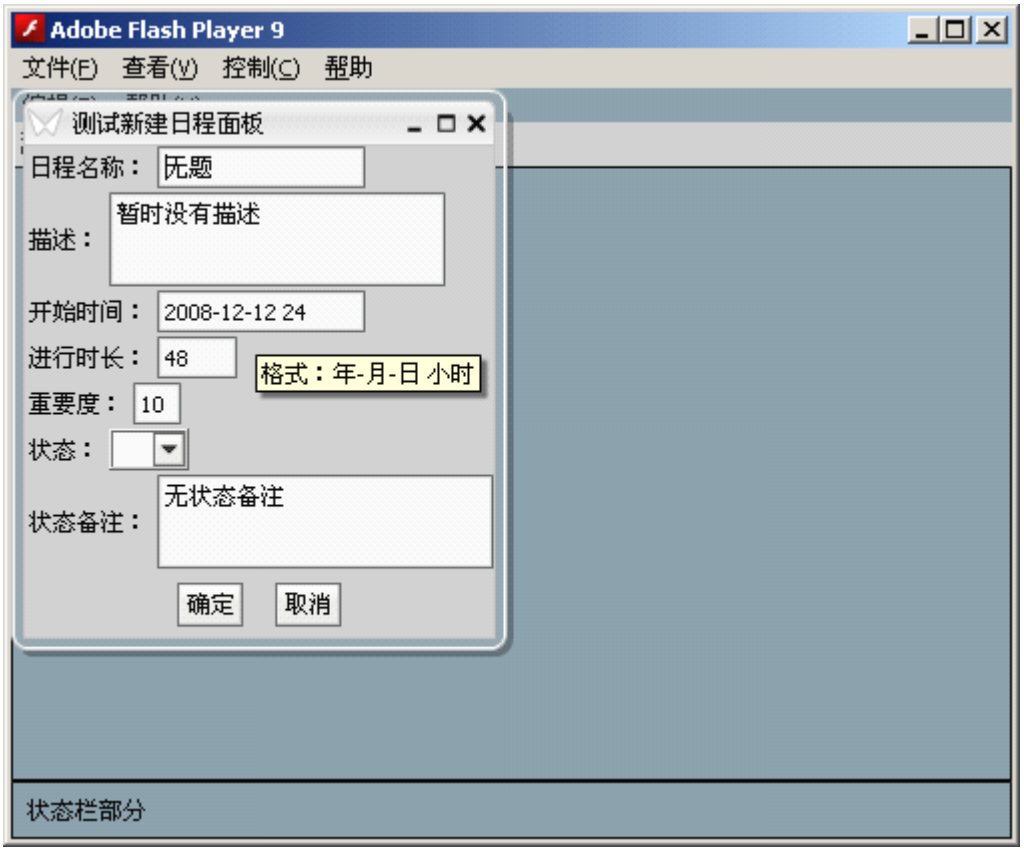
```
package book.scheme{

import org.aswing.Component;
import org.aswing.Container;
import org.aswing.FlowLayout;
import org.aswing.JButton;
import org.aswing.JComboBox;
import org.aswing.JLabel;
import org.aswing.JPanel;
import org.aswing.JTextArea;
import org.aswing.JTextField;
import org.aswing.SoftBoxLayout;

/**
 * 新建日程面板
 */
public class CreateTaskPane extends JPanel{

    private var nameText:JTextField;
    private var descriptionText:JTextArea;
    private var startTimeText:JTextField;
    private var processTimeText:JTextField;
    private var importanceText:JTextField;
    private var statusCombo:JComboBox;
    private var statusComment:JTextArea;

    private var okButton:JButton;
    private var cancelButton:JButton;

    public function CreateTaskPane(){
        super();

        //创建供用户编辑数据的组件
        nameText = new JTextField("无题", 12);
        descriptionText = new JTextArea("暂时没有描述", 3, 20);
        startTimeText = new JTextField("2008-12-12 24", 12);
        startTimeText.setRestrict("0123456789 \\-");
        startTimeText.setMaxChars(13);
        processTimeText = new JTextField("48", 4);
        processTimeText.setRestrict("0123456789");
        importanceText = new JTextField("10", 2);
        importanceText.setRestrict("0123456789");
        importanceText.setMaxChars(2);
        statusCombo = new JComboBox();
```

```
        statusComment = new JTextArea("无状态备注", 3, 20);
        okButton = new JButton("确定");
        cancelButton = new JButton("取消");
        //布局并加入标头和提示标签
        setLayout(new SoftBoxLayout(SoftBoxLayout.Y_AXIS, 2));
        append(labelHold(nameText, "日程名称："));
        append(labelHold(descriptionText, "描述："));
        append(labelHold(startTimeText, "开始时间：", "格式：年-月-日 小时
"));
        append(labelHold(processTimeText, "进行时长：", "单位小时"));
        append(labelHold(importanceText, "重要度：", "0到10"));
        append(labelHold(statusCombo, "状态："));
        append(labelHold(statusComment, "状态备注："));
        var buttonPane:JPanel = new JPanel(new
FlowLayout(FlowLayout.CENTER, 16, 5));
        buttonPane.appendAll(okButton, cancelButton);
        append(buttonPane);
    }

    //创建一个容器，左边显示一个指定字符串text的标签，右边显示指定组件c，
    //通过toolTip指定需要显示的提示文字，如果不需要，则传入null
    private function labelHold(c:Component, text:String,
toolTip:String=null):Container{
        var panel:JPanel = new JPanel(new FlowLayout(FlowLayout.LEFT,
2, 0, false));
        var label:JLabel = new JLabel(text);
        panel.appendAll(label, c);
        if(toolTip != null){
            panel.setToolTipText(toolTip);
        }
        return panel;
    }
}
}
```

为了快速验证这个面板的效果，我们在主类 SchemeManager 中加入一个测试方法如下：

```
    private function testCreateTaskPane():void{
        var frame:JFrame = new JFrame(null, "测试新建日程面板");
        var pane:CreateTaskPane = new CreateTaskPane();
        frame.setContentPane(pane);
        frame.pack();
        frame.show();
    }
```

在 SchemeManager 的构造函数中调用此方法，编译并运行程序，我们将看到如下画面：



（图 8）

看来很简单，新建日程的界面算是完工了（虽然布局算不上美观）。CreateTaskPane 类用到了新的组件单行文本框 JTextField 和多行文本框 JTextArea，还有下拉列表 JComboBox。testCreateTaskPane 方法里用到了 JFrame（本节暂时不做讲解，为避免本节内容过多，后面章节再次用到时才做讲解）。JTextField 和 JTextArea 拥有共同的父类 JTextComponent，分别用来显示和输入单行和多行文字。JComboBox 是一个拥有一个文本和箭头按钮的组件，默认状态下显示出选中的内容，当点击下拉箭头时，弹出一个内容列表供选择。

此面板的布局方法，我们将在下面的知识点讲解后再做分析。

本节知识点：

本节的重点是布局原理和常用布局管理器。下面我们会先讲解本节用到的新组件，然后重点详细的讲解布局内容。

JTextComponent：文本框的基类，通常使用它的子类而不是直接使用此类创建界面，它提供了文本框组件的所有共有特性。比如本节使用到的 setRestrict 方法，设置输入限制（限制格式请参见 flash.text.TextField 类的 restric 属性），setMaxChars 方法，设置输入字符数限制（参见 flash.text.TextField 类的 maxChars 属性）。FlashPlayer 类库中的 flash.text.TextField 的大多数属性和方法，此类都有提供。实际上，JTextComponent 的内部正是通过一个 TextField 来实现的，并且通过 getTextField()方法，你还可以直接访问到这个 TextField 实例。

JTextField：一个只提供单行输入/显示的文本框组件。构造函数形式为*JTextField(text:String="", columns:int=0)*，第一个参数指定要显示的字符串内容，第二个参数指定文本框期望宽度（字符数单位，对应于实际期望的像素宽度，这是一个相对值，它相对于当前字体下“w”字符的宽度，也就是说，如果指定 columns 为 n，那么文本框期望的宽度将是刚好能容下 n 个“w”字符那么宽）。对于字符内容，还可以通过 setText 方法来设置。

JTextArea：一个提供多行显示/输入的文本组件，构造函数形式为*JTextArea(text:String="", rows:int=0, columns:int=0)*。text 和 columns 属性意义和 JTextField 相同，rows 属性与 columns 属性类似，它作用于文本框的期望高度。

JComboBox：本节我们并没有真正使用这个组件，我们将在稍后再次给此组件添加数据时讲解。

布局原理：读者可能在内心早已有所疑问，本章从开始到现在，从来没有调用过一次组件的 setLocation 或 setSize 方法设置它们的位置和大小，那么这些组件是如何缩放和定位的呢。这一节笔者终于有机会详细介绍这个内容，这是掌握 AsWing 所需理解的最重要的知识点之一。

1. **容器与布局管理器**：在 AsWing 框架中，一个界面中的组件（Component）层次结构属于组合模式，即容器（Container）包含组件，而容器也是组件（组件的子类），所以容器可以包含组件也可以包含另一个容器，层层叠叠。每个组件都需要有一个显示范围（bounds——包含 location 和 size），才能被肉眼所见。通常，放置于容器中的组件的显示范围由容器来决定，而容器本身不提供显示范围的计算方法，这一工作被提取出来交给容器所使用的布局管理器（LayoutManager）来实现。即，容器本身只包含组件，组件的显示范围由布局管理器来决定。
2. **布局管理器**：布局管理器是指实现了 LayoutManager 接口的对象，一般来说所有容器都必须拥有布局管理器，否则无法将所拥有的组件进行布局。AsWing 类库中自带了一些常用的布局管理器实现，比如 FlowLayout, BorderLayout, SoftBoxLayout, BoxLayout, GridLayout 等，这些布局管理器的特点我们稍后将一一进行介绍。开发者也可以通过自己编写 LayoutManager 接口的实现对象来创造自己特有的布局管理器。
3. **布局管理器的准则**：布局管理器主要需要做两个工作，第一个是计算容器的期望大小（preferredLayoutSize），第二个是对容器中的组件设置显示范围（通过调用组件的 setComBounds 或者调用 setLocation+setSize）。对于第一个工作，由于容器的期望大小应该是合理显示所有包含组件后需要的大小（组件的期望大小可通过 getPreferredSize 来获得），比如，一个横向排列的布局，容器所期望的大小，高度则应该是所有组件中最高的那个组件的高度，宽度则应该是所有组件的宽度之和。第二个工作，对所有包含的组件依次设置显示范围，按照自己的策略结合组件期望大小来进行，又比如一个横向排列的布局，第一个组件的 x 坐标应该是 0，第二个组件应该在第一个组件靠右的位置，至于组件的大小，高度可以遵从组件的期望高度，也可以直接设置为容器高度（满填充），宽度可以遵从组件的期望宽度也可以各组件平分容器的宽度，具体如何，视布局管理器本身的设计目的而定。**归根结底，布局管理器通常遵循如下准则：始终以容器所直接包含的组件的期望大小为依据，来进行容器期望大小的计算和组件显示范围的设置。**
4. **期望大小**：所有组件，包括容器，都可以直接通过 getPreferredSize 方法获得它们的期望大小。其实，上面我们已得知，在 getPreferredSize 的内部实现上，期望大小是

由布局管理器计算出，而此计算是需要以所包含组件的期望大小为依据。那么对于一个不是容器的组件，它怎样计算自己的期望大小呢？组件的期望大小由组件自己进行计算，也就是说，JButton 自己需要计算自己的期望大小，JLabel 也是如此，使用者无需关心它们内部的算法是如何实现的。如果使用者需要强制指定一个大小作为期望大小，可以调用 setPreferredSize 方法来设置一个值，设置了一个定值之后，组件将不会自己进行计算，永远返回这个指定的值，如果调用 setPreferredSize 设置一个 null 值，组件则返回自己计算大小的策略（容器也是如此）。对于组件开发者，在开发一个新组件时，如果这个组件不是容器，那么也要自己覆盖 getPreferredSize 方法来实现合理的计算，通常，按照正好能显示出组件内容为准则来计算大小。

5. **最大最小尺寸**：和期望大小类似，组件也需要提供最大和最小尺寸，通常的实现，最大值提供一个足够大的值即可（比如 10000*10000），最小值为 0*0 即可。在 AsWing 框架中，组件的最大最小尺寸并没有期望尺寸那么重要，通常可以忽略，采用通用实现即可。（很少情况下，会用到这两个值，在 Resizer 的实现中，这两个值被用来判断可缩放到的最大最小尺寸，一些 LookAndFeel 的 UI 实现也许会考虑最小尺寸，当组件被设置为小于最小尺寸时，仍按最小尺寸来绘制组件，但组件由于有范围遮罩，所以仍能以设置的大小显示，超出的部分则不可见。）

如何使用布局管理器：*（这部分内容改编自《AsWing 布局管理器入门》，作者 harry，地址：http://cn.aswing.org/?p=13）*

前面说了每个容器中都有一个布局管理器，所以当我们要为一个容器指定一个布局的时候调用容器的 setLayout 方法即可，由于此项机制，我们可以在运行时动态改变容器的布局。

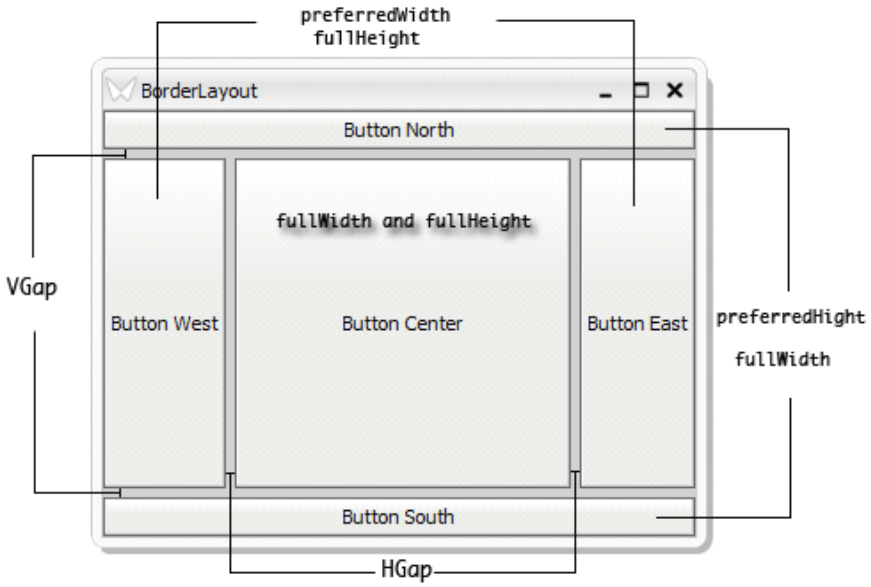
当需要将某个组件加入到容器中时，不是用 addChild，而是要用 append 或 insert 方法，append 将组件追加到容器尾部，instert 将组建添加到容器的指定位置。append 和 insert 调用后，方法内部会同时调用容器中布局对象的 addLayoutComponent 方法，将组件加入到布局管理器中。这两个方法的最后一个参数为可选参数，这是对于此组件的布局约束——如果指定了该参数，它最终会被传递给布局管理器的 addLayoutComponent 方法。如果使用 addChild 方法添加组件，则组件会被视为普通显示元件而得不到布局管理。

大部分布局管理器会用到容器内组件的尺寸信息，在进行排列时会作为逻辑判断与最终设置组件的实际尺寸，这使得开发者可以给组件提供尺寸信息以达到想要的效果。一般情况下布局管理器都会优先使用组件的 preferredSize，如果有些布局明确说明以 preferredSize 显示组件尺寸，那么为组件设置的 preferredSize 就是最终显示出来的尺寸。当然也有可能有布局使用绝对尺寸来显示组件，比如 EmptyLayout，要指定布局中组件的尺寸就是用 setSize。这些在不同的布局中都会有各自不同的情况，在使用具体的布局管理器时，需要先阅读该布局的描述文档，然后决定用何种方式来控制容器内组件的尺寸。

有时候对容器内的组件期望尺寸信息进行了改变，却没有立即看到效果，那可能是由于布局管理器缓存了布局信息，这时候我们就需要使布局失效，迫使他在下一次重绘的时候重新计算排列容器内的组件。比较常用的方法是调用已发生改变的组件的 revalidate 方法，这样会使调用该方法的组件与该组件外层的所有容器都标记为需要重新布局。

常用布局管理器介绍：*（这部分内容改编自《AsWing 布局管理器入门》，作者 harry，地址：http://cn.aswing.org/?p=13）*

BorderLayout：只负责里容器中 5 个组件的排列方式，这五个组件的位置分别位于 东、南、西、北、中 方向，请看下图：



（图 9）

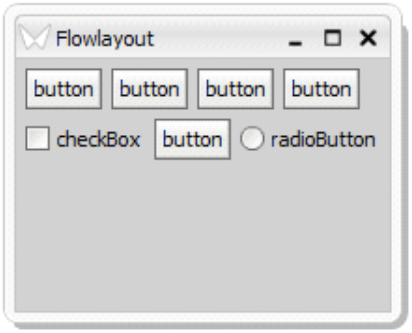
上图的容器中（JFrame 的 ContentPane）有 5 个按钮（JButton）组件，分别位于 North, South, West, East 和 Center 。要为容器指定一个 BorderLayout，可以使用 *container.setLayout(new BorderLayout());*

BorderLayout 的构造函数可传入两个参数，即 hgap 和 vgap，vgap 是纵向组件之间的间隔，hgap 是横向组件之间的间隔，可以用 setVgap 和 setHgap 来设置。

位于 North 和 South 位置的组件高度为他们的 preferredHeight，而宽度就是容器的宽度，图中的 fullWidth 和 fullHeight 指可用的最大宽度和高度。位于 West 和 East 位置的组件的宽度被设置为他们的 preferredWidth，高度为 fullHeigth。Center 位置即中间的那个组件的尺寸将被设为剩余区域的大小。

要在向容器中添加组件时指定组件在容器中所处的位置，就可以用 append 方法中的第二个参数来制定约束。如： *container.append(component, BorderLayout.NORTH);*，它会将 component 添加到容器的 North 位置（北方）。记住，虽然我们调用的是容器的 append 方法，但是最终决定如何排列组件的是容器中的布局管理器，容器会调用其布局管理器的相应方法，传入组件和约束，布局管理器会根据约束来排列组件并且设置组件的尺寸。

FlowLayout：一种比较简单的布局方式，它会将所有组件排列成一行，以组件的 preferredSize 显示，一般情况下，如果一行显示不了所有的组件，会自动换到下一行显示。如图：



（图 10）

这是一个典型的 FlowLayout 布局，容器内的组件都以 preferredSize 显示，由于在一行中不够显示所有容器，所以布局管理器将剩余的组件换到了第二行显示。

FlowLayout 的构造函数可以接受 4 个参数：align, hgap, vgap, margin。

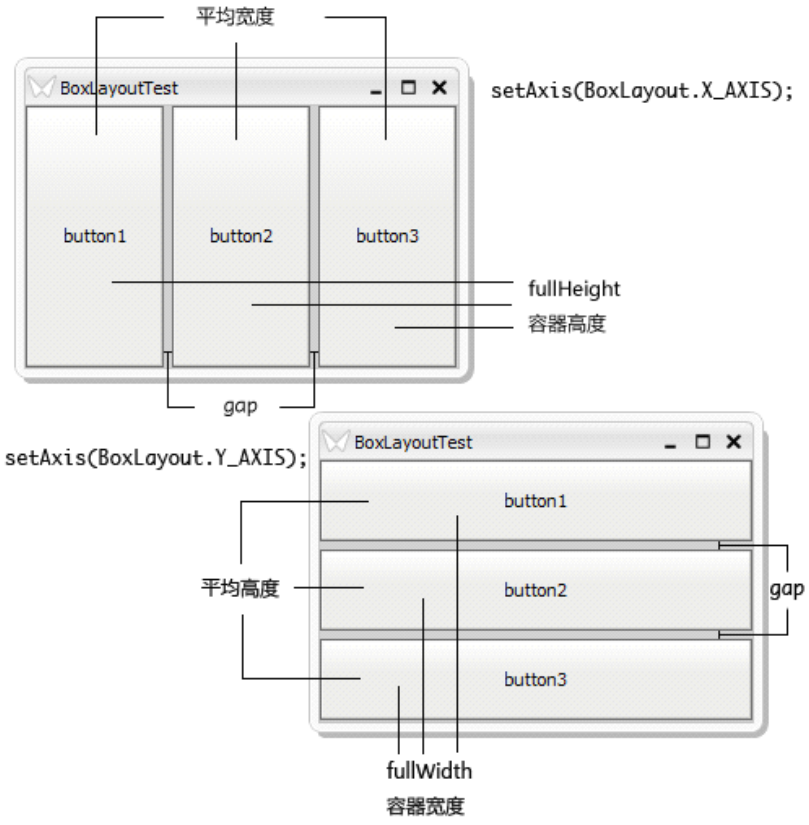
align 表示对齐方式，可以传入 FlowLayout.LEFT, FlowLayout.RIGHT, 或 FlowLayout.CENTER，分别表示左对齐，居中对齐和右对齐。

组件之间的间隔和 BorderLayout 一样，也是通过 hgap 和 vgap 定义。

margin 为一个布尔参数，指示是否将间隔作用与容器的四周，默认为 true，如果设置为 false 的话，组件将紧贴容器的边框。

要向使用了 FlowLayout 的容器中添加组件，只需要简单地调用容器的 append 或 insert 方法即可，无需指定约束，因为 FlowLayout 对于组件的排列是一个挨一个的，没有特殊的约束定义。

BoxLayout：对容器中的组件进行同一方向上的平均排列，纵向或者横向：



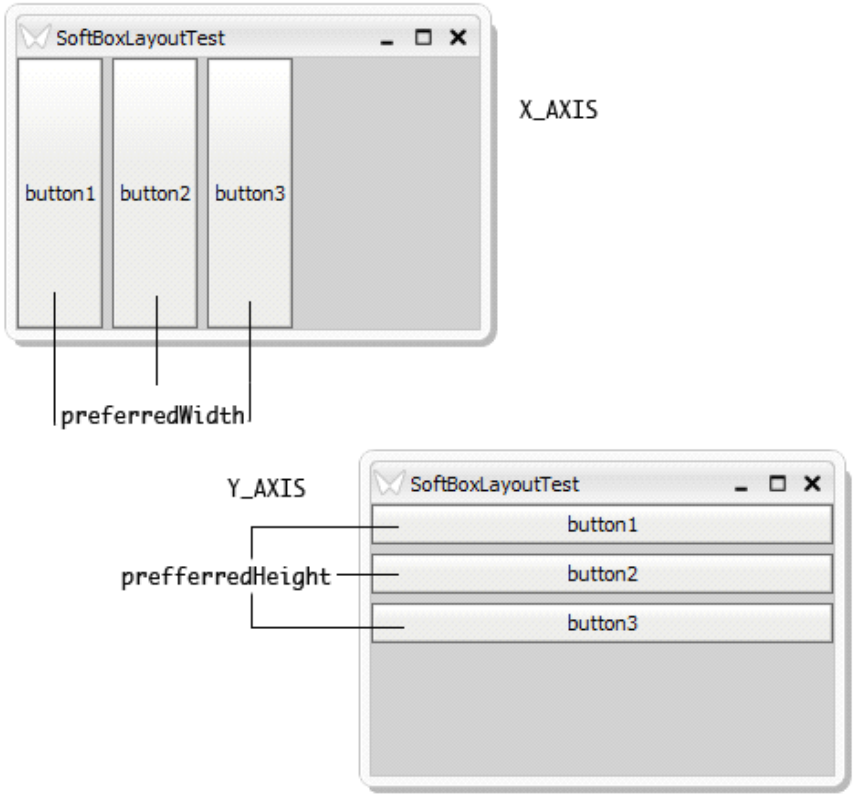
（图 11）

BoxLayout 的构造函数有两个参数，axis 和 gap。axis 用于指示容器中组件的排列方向，一种有 2 种，BoxLayout.X_AXIS 和 BoxLayout.Y_AXIS ，即横向与纵向。gap 就是制定组件之间的间隔，因为 BoxLayout 排列的组件不是横向就是纵向，不会出现横向纵向同时出现的情况，所以只要一个 gap 参数即可。

在横向排列的情况下，容器内组件的宽度是容器的宽度减去所有间隙的和然后平均分配得到的。组件的高度则以撑满容器为标准。

纵向排列的方式雷同，只是将 X 轴与 Y 轴的排列方式进行了对调。

SoftBoxLayout：SoftBoxLayout 与 BoxLayout 非常相似，使用上也几乎一样，但是，对于组件尺寸的设置有所不同。当布局被设为横向排列时，容器内的组件宽度会被设置成各组件的 preferredWidth，高度为容器高度。当布局被设为纵向排列时，情况正好相反，组件的宽度被设为容器宽度，高度为各组件的 preferredHeight。



（图 12）

SoftBoxLayout 的构造函数接受 3 个参数，axis, gap, align。前两个参数的作用与 BoxLayout 构造函数的参数作用相同，即排列方向和组件间隔。第三个参数为容器内组件的对齐方式，当排列为横向排列时，可以设为 AsWingConstants.LEFT 、AsWingConstants.CENTER 、AsWingConstants.RIGHT，即左对齐、居中对齐、右对齐。

如果排列方式为纵向排列，可以设为 AsWingConstants.TOP、AsWingConstants.CENTER、AsWingConstants.BOTTOM，即上对齐、居中对齐、下对齐。

其他布局方式：上面介绍了 4 种比较常用的布局方式，通过这 4 种布局方式的组合已经

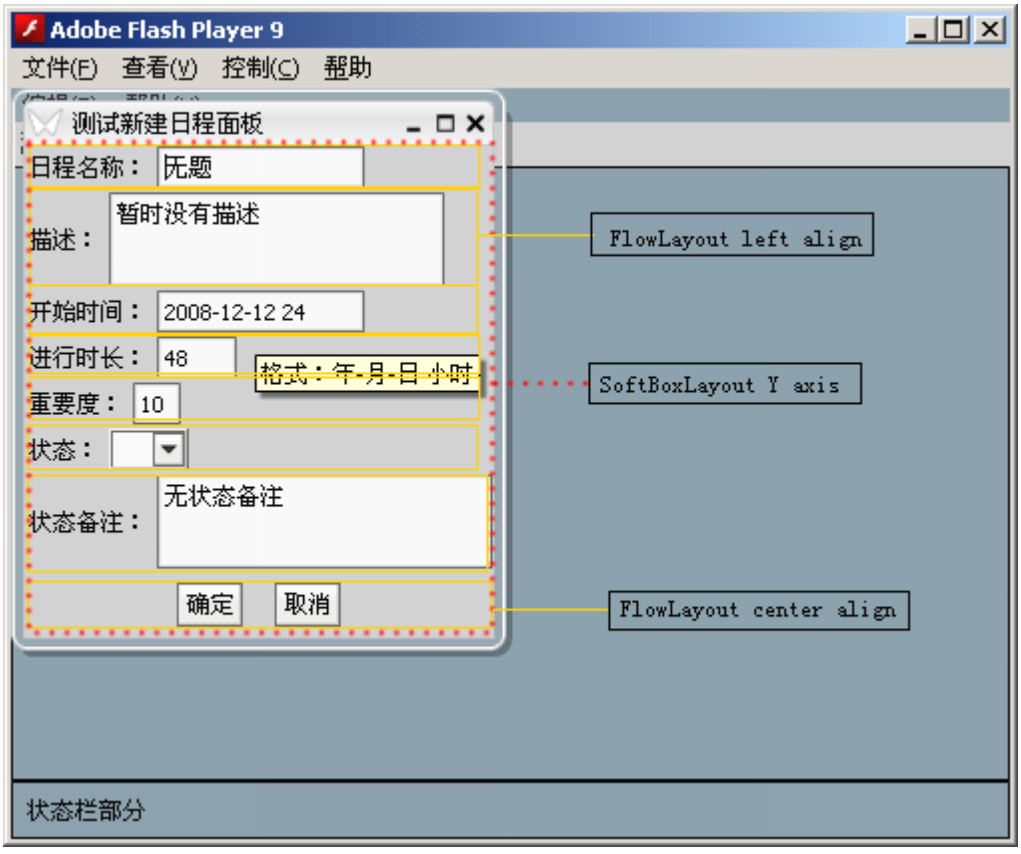
可以满足大多数界面的布局工作。当然 AsWing 中的布局方式还不止这些，相信读者只要掌握了这 4 种布局，再学习新的布局方式将不成问题。

上面的这些布局方式，都是对容器内的组件按一定规则进行排列和调整尺寸，如果要完全手动调整，进行绝对定位的话，可以使用 EmptyLayout。

EmtpyLayout 是对 LayoutManager 接口的空实现，只是简单的实现了接口方法，没有具体算法，所以扔到 EmptyLayout 容器中的组件都可以直接 setSize，和 setLocation，设置多少，运行时看到的就是多少。但是注意，**由于是空实现，所以放到容器中的组件没有默认尺寸，如果不对其进行 setSize，是看不到的**。

另外还有 GridLayout——对容器中的组件进行网格式的排列；CenterLayout——只管里容器中的一个组件，使其在容器中居中；FlowWrapLayout——FlowLayout 的改进版，可以指定每行的最大宽度，如果大于指定宽度则开始换行。如此等等，读者可以通过 AsWing 的 API 文档进行查看，如果觉得没有自己需要的，还可以自行实现 LayoutManager 接口，创建自己的布局方式。

新建日程界面布局解析：通过对布局的学习和了解，结合代码，我们现在可以分析出图 8 所示界面的布局结构，如下图：



（图 13）

每个属性条目是由两个组件构成，装载它们的是一个 JPanel，使用 FlowLayout 布局，对齐方式为靠左（详见 labelHold 方法）。最下面的两个按钮，使用的也是FlowLayout 布局，居中对齐。条目之间，采用的是 SoftBoxLayout，Y 方向排列。由此可见，一个复杂的界面布局，可以通过多层容器+布局管理器嵌套的方式来实现。理论上大多数应用程序界面布局

都可以通过 AsWing 自带的几种常用布局管理器组合嵌套来实现，但有些情况还是自己写新的布局管理器更合适一些。如果情况实在太过复杂，可以考虑使用 EmptyLayout，然后对每个组件手动设置绝对的大小和位置信息，达到绝对自由的效果。

2.3.2 通过界面创建数据（控制器，JFrame，JOptionPane）

数据结构（Model）和显示界面（View）已经完成了，这一步我们需要创建控制器（Controller）。

新建日程控制器的作用即是提供弹出界面的方法，用户点击确定后根据填入的数据生成日程对象。由此，我们可以编写如下代码：

CreateTaskController.as

```
package book.scheme{

import flash.events.Event;

import org.aswing.JFrame;
import org.aswing.JOptionPane;

/**
 * 新建日程控制器
 */
public class CreateTaskController{

    private var handler:Function;
    private var dialog:JFrame;
    private var pane:CreateTaskPane;

    /**
     * 构造新建日程控制器
     * @param handler 新建结果处理器，格式 handler(task:Task)
     *      如果task为null代表取消了创建
     */
    public function CreateTaskController(handler:Function){
        this.handler = handler;
        pane = new CreateTaskPane();
        pane.getStatusCombo().setListData([
            TaskStatus.PLAN,
            TaskStatus.PROCESSING,
            TaskStatus.CANCEL,
            TaskStatus.FINISHED,
```

```
            TaskStatus.INTERRUPTED,
            TaskStatus.MISS
        ]);
        pane.getStatusCombo().setSelectedIndex(0);

        dialog = new JFrame(null, "新建日程", true);
        dialog.setContentPane(pane);

        pane.getOkButton().addActionListener(__ok);
        pane.getCancelButton().addActionListener(__cancel);
    }

    private function __ok(e:Event):void{
        var task:Task = new Task();
        task.description = pane.getDescriptionText().getText();
        task.importance = Math.min(10,
            int(pane.getImportanceText().getText()));
        task.name = pane.getNameText().getText();
        task.processTime = int(pane.getProcessTimeText().getText());
        try{
            task.startTime = Utils.stringToDate(
                pane.getStartTimeText().getText());
        }catch(error:Error){
            JOptionPane.showMessageDialog("提示", error.message);
            return;
        }
        task.status = pane.getStatusCombo().getSelectedItem();
        task.statusComment = pane.getStatusComment().getText();
        dialog.dispose();
        handler(task);
    }

    private function __cancel(e:Event):void{
        dialog.dispose();
        handler(null);
    }

    public function show():void{
        dialog.pack();
        dialog.show();
    }
}
}
```

同时编写工具类 Utils 用于提供项目通用的日期事件与字符串之间的转换：

Utils.as

```
package book.scheme{

/**
 * 工具类
 */
public class Utils{

    /**
     * 把一个year-month-day hour格式（比如2008-8-7 10）的字符串转换为Date
     * @throws Error 如果字符串格式不正确，则抛出Error
     */
    public static function stringToDate(str:String):Date{
        var strs:Array = str.split(" ");
        if(strs.length != 2){
            throw new Error("日期格式不正确（year-month-day hour）");
            return null;
        }
        var dateStr:String = strs[0];
        var timeStr:String = strs[1];
        var dateSpl:Array = dateStr.split("-");
        if(dateSpl.length != 3){
            throw new Error("日期格式不正确（year-month-day hour）");
            return null;
        }
        var year:int = int(dateSpl[0]);
        var month:int = int(dateSpl[1])-1;
        var day:int = int(dateSpl[2]);
        var hour:int = int(timeStr);
        return new Date(year, month, day, hour);
    }

    /**
     * 把一个Date转换为year-month-day hour格式（比如2008-8-7 10）的字符串
     */
    public static function dateToString(date:Date):String{
        return date.getFullYear()
                + "-" +( date.getMonth()+1)
                + "-" + date.getDay()
                + " " + date.getHours();
    }

}
```

```
}
```

为了便于讲解，控制器我们尽量设计得比较简单，对于用户输入的判断也只是做了简单的检查。此控制器对外的接口只有两个，一个是构造函数，一个是 show 函数。构造函数要求使用者传入一个处理器函数，当用户完成或取消新建时，处理器函数会得到调用，并被传入创建好的日程对象。show 函数只是简单的使新建界面显示出来。为了尽快测试效果，我们修改 SchemeManager 的 testCreateTaskPane 方法为：

```
    private function testCreateTaskPane():void{
        new CreateTaskController(function(task:Task):void{
            trace("Created Task : " + task);
        }).show();
    }
```

编译并运行程序，直接点击“确定”得到默认设置的 task 对象，控制台看到输出为：

Created Task : Task[name:无题, description:暂时没有描述, startTime:Sat Dec 13 00:00:00 GMT+0800 2008, processTime:48, importance:10, status:计划, statusComment:无状态备注]

可见 Task 对象已经成功创建出来了，如果点击“取消”，得到如下输出：

Created Task : null

代表用户取消了新建。

在内部实现上，__ok 方法是对用户点击“确定”按钮的处理，根据界面各组件的输入的数据来创建 Task 对象(本节还对 CreateTaskPane 添加了一系列 get 方法让用户获得输入组件，具体代码没有列出，读者可查阅本大节的最终源代码)，对于字符串类型的属性，JTextField/JTextArea 的 getText 方法直接获得最终的值，对于整数类型，我们用 int(String)方式进行转换，对于日期类型，我们通过 Utils.stringToDate 方法进行转换，此方法的实现稍微复杂一些，并且我们进行了错误格式的检测，如果遇到格式不正确，则抛出异常，因此在__ok 方法中我们必须捕获可能的异常，当异常发生时，弹出提示框，并终止 Task 的创建。Utils.dateToString 方法暂时没有使用到，这里先实现出来留作以后使用。

本节知识点：

控制器（Controller）的实现：本节演示了一个典型的控制器实现方法。通常，控制器负责提供公共接口给外部，以使得外部可以通过控制器实现数据的创建、更改、删除等功能，控制器通过显示（View）来操作数据（Model），但并不透露内部细节。通过 handler 处理器的方式，使得外部更容易操作——只需传入一个处理器函数即可得到结果。

JFrame：带标题栏的标准窗体。它是 JWindow 的子类，由一个默认的 FrameTitleBar 实现（JFrameTitleBar）和 contentPane（Container）构成。通常情况下，使用默认的 FrameTitleBar 即可（即不需要另行调用 setFrameTitleBar 来设置），contentPane 是一个空的容器，通常需要调用 setContentPane 或者 getContentPane().append 来设置内容面板。本节调用 setContentPane 把一个 CreateTaskPane 实例设置为内容面板。JFrame 的构造函数形式为 *JFrame(owner:*=null, title:String="", modal:Boolean=false)*第一个参数指定它的所有者，可以是一个 JPopup 或者 DisplayObjectContainer，默认 null 代表使用 AsWingManager.getRoot()返回的值做拥有者。通常，采用 null 即可，如果指定特定的拥有者，那么这个窗体将创建于拥有者原件之中，这常常用于窗口的子窗口或者给特定元件容器中创建窗口来实现窗口的固

定深度关系，第二个参数指定窗体的标题文字，第三个参数指定窗口是否为独占的，独占是指只有当前窗口能获得鼠标和键盘焦点，此窗口外的其他内容无法相应用户的操作，这对于一些需要强制用户完成一定输入再恢复其他操作有非常重要的意义。本节中在调用 JFrame.show 显示窗体前，调用了 pack 方法，pack 方法继承自 Component 类，它的作用是采用组件的期望大小来设置大小，由于窗体是根组件（即它是容器的最顶层，没有其他容器包含它），所以 pack 方法常常用于窗体（或者 JPopup，JWindow 等其他跟组件）的大小设置。

JOptionPane：选项提示组件（在一些组件库中，此类通常被命名为 Alert）。此组件提供一系列静态方法，用于弹出选项提示窗体。 showMessageDialog 用于弹出一个含有标题，文字标签，及选项按钮的窗体。showInputDialog 在 showMessageDialog 的基础上多了一个输入框，让用户输入一个字符串。本节的例子，当用户输入的日期格式不正确时，弹出一个提示框告诉用户，此处便是使用的 showMessageDialog 方法。此类的各参数及其含义具体请参见 api 文档。

2.3.3 用 Form 再造新建界面（Form 布局）

图 8 的新建界面，对于一个表单式的界面，显得不够整齐美观。虽然通过对各组件的期望尺寸进行统一调整可以达到整齐的目的，但是实现起来过于麻烦，而且需要精心调整才可以达到，读者可以自行实验一下。本章将使用 AsWing 1.3 版本引入的专门用于表单布局的容器 Form 来为大家重新构建新建界面。

为了实现表单的整齐布局，同时又要灵活满足复杂表单的要求，Form 的实现便显得略为复杂，不过接口的设计却很简洁，因此使用起来非常方便。采用 Form 重新布局新建界面的代码如下：

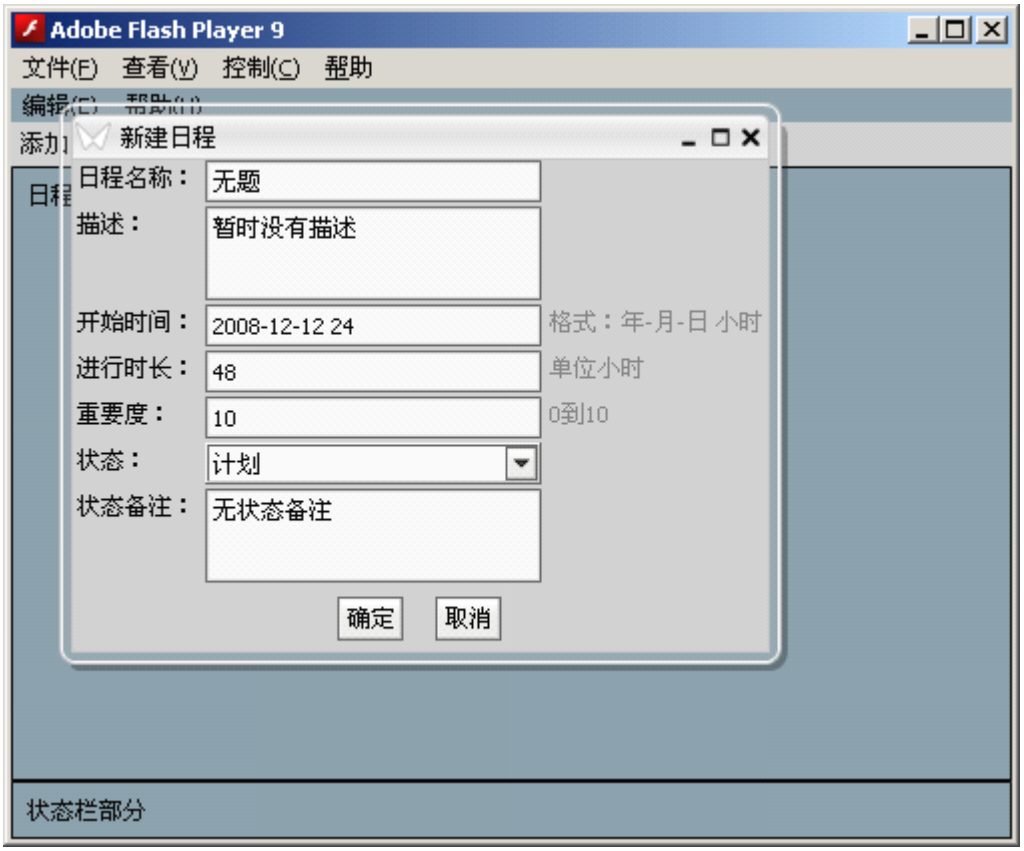
CreateTaskPane2.as

```
package book.scheme{

import org.aswing.*;
import org.aswing.ext.Form;

/**
 * 采用Form布局的新建日程面板
 */
public class CreateTaskPane2 extends Form{

    private var nameText:JTextField;
    private var descriptionText:JTextArea;
    private var startTimeText:JTextField;
    private var processTimeText:JTextField;
    private var importanceText:JTextField;
```

```
    private var statusCombo:JComboBox;
    private var statusComment:JTextArea;

    private var okButton:JButton;
    private var cancelButton:JButton;

    public function CreateTaskPane2(){
        super();
        //创建供用户编辑数据的组件
        nameText = new JTextField("无题", 12);
        descriptionText = new JTextArea("暂时没有描述", 3, 20);
        startTimeText = new JTextField("2008-12-12 24", 12);
        startTimeText.setRestrict("0123456789 \\-");
        startTimeText.setMaxChars(13);
        processTimeText = new JTextField("48", 4);
        processTimeText.setRestrict("0123456789");
        processTimeText.setMaxChars(3);
        importanceText = new JTextField("10", 2);
        importanceText.setRestrict("0123456789");
        importanceText.setMaxChars(2);
        statusCombo = new JComboBox();
        statusCombo.setPreferredWidth(100);
        statusComment = new JTextArea("无状态备注", 3, 20);
        okButton = new JButton("确定");
        cancelButton = new JButton("取消");
        //布局并加入标头和提示标签
        addLabelRow(nameText, "日程名称：");
        addLabelRow(descriptionText, "描述：");
        addLabelRow(startTimeText, "开始时间：", "格式：年-月-日 小时");
        addLabelRow(processTimeText, "进行时长：", "单位小时");
        addLabelRow(importanceText, "重要度：", "0到10");
        addLabelRow(statusCombo, "状态：");
        addLabelRow(statusComment, "状态备注：");

        var buttonPane:JPanel = new JPanel(new
FlowLayout(FlowLayout.CENTER, 16, 5));
        buttonPane.appendAll(okButton, cancelButton);
        append(buttonPane);
    }

    //添加一个FormRow，text为第一格标签，c为第二格的组件，toolTip不为null的话
为第三格标签
    private function addLabelRow(c:Component, text:String,
toolTip:String=null):void{
```

```
        var label:JLabel = new JLabel(text, null, JLabel.LEFT);
        var tip:JLabel = null;
        if(toolTip){
            tip = new JLabel(toolTip, null, JLabel.LEFT);
            tip.setForeground(ASColor.GRAY);
        }
        addRow(label, c, tip);
    }
    //……
    //这里省略组件的get方法
}
}
```

替换 CreateTaskController 代码中的 CreateTaskPane 为 CreateTaskPane2，编译并运行，将看到如下界面：



（图 14）

由此可见，采用 Form 布局的界面，非常整齐美观。Form 最主要的接口是 addRow，它实际上是根据传入的组件创建一个 FormRow，FormRow 是 Form 所包含和管理的子容器，代表一行。

要准确和灵活的使用 Form，必须得了解一下它的工作原理：

Form 的意图在于提供一个类似表格布局工具。在单纯的表格上，给予不同的行可以自适应不同的高度，一个列上面的所有组件拥有相同的高度，但是一个组件可以占用 1 个以上的列，也允许某一行中的列空余。假设我们的 Form 会布局为一个 4*4 的表格：

00	01	02	03

10	11	12	13
20	21	22	23
30	31	32	33

通过 addRow 方法，我们可以依次向任何一个格子加入一个组件，第一次调用 addRow 会创建出第一行，addRow 方法中参数的数量代表这一列需要占用的列数。依次调用四次 addRow，最多参数的那次调用传了 4 个参数，就会形成如上 4*4 的表格。加入我们调用这几个语句：

```
addRow(label1, text1_little_long, label4);

addRow(label2, text2_high, label5);

addRow(label3_a_long_label_here, text3, combo1, label6);
addRow(null, button1, button2);
```

则会得到如下布局：

label1	text1_little_long	label4	
label2	text2_high	label5	
label3_a_long_label_here	text3	combo1	label6
	button1	button2	

注意表格边框不会在 Form 组件中出现，这里为了方便观察各组件所占格的范围，所以布局图明确的绘制了边框。可以发现，每行所有组件的高度自适应到能显示行上所有组件的高度，每列所有组件的宽度自适应到能显示列上所有组件的宽度，正因为如此特性，保证了既能正确显示出所有组件，又能整齐的布局。另外，没有传入组件的格子，保持空白（表格中白色格子部分），这满足了需要空白的需求。

上面的例子，还只是非常简单的应用，前面我们还说过 Form 支持一个组件占用多个列，请看下例：

```
addRow(label1, text1_little_long, a_button_here, label4);
var checkPanel = new JPanel(new FlowLayout());
checkPanel.appendAll(checkbox1, checkbox2, checkbox3);
addRow(label2, checkPanel, checkPanel, label5);
var lc = a_very_long_component_want_sit_here;
addRow(lc, lc, lc, label6);
addRow(button1, null, null, button2);
```

则会得到如下布局：

label1	text1_little_long	a_button_here	label4
label2	checkbox1 checkbox2 checkbox3		label5
a_very_long_component_want_sit_here			label6
button1			button2

这个布局的第一个特殊之处在于 checkbox1，checkbox2 和 checkbox3 并没有整齐分处与不同单元格，而是前后紧挨着位于 11 和 12 单元格总范围内。实现方法为，把 checkbox1、checkbox2 和 checkbox3 用一个 FlowLayout 布局于一个 JPanel 容器中，然后把这个 JPanel 放置于 11 和 12 单元格总范围内，相当于合并了 11 和 12 单元格。如上代码，连续在第二个和第三个参数都传入 checkPanel 即代表 checkPanel 会占用此行第二和第三单元格合并的范围。同理，a_very_long_component_want_sit_here 通过前三个参数都传入它，使得它坐落于第三行的前三格合并的范围中。

当 Form 中有部分组件尺寸太大，需要占用多个列时（如上述的 a_very_long_component_want_sit_here），或者当有几个小尺寸组件，但又不想让它们单独占用单元格从而影响整体布局规划时（如上述的 checkbox1-3），此特性非常有用。

另外，Form 除了可以使用 addRow 添加组件，也可以使用容器方法 append。当 append 传入的是一个 FormRow 实例时，与调用 addRow 实际上是一样，因为 addRow 的内部实现上，就是创建一个 FormRow 然后 append。当append 传入的是一个非 FormRow 实例时，Form 对于这个传入的组件，表现为类似 SoftBoxLayout 的布局行为，当需要添加一个组件占用整行时，此方式非常方便，因为它不会影响 Form 的列数（相反，addRow 传入的参数个数，直接影响到 Form 采用的列数——最大数量参数即为最终形成的列数）。本节实现的 CreateTaskPane2 对于“确定”和“取消”按钮的布局，便是通过此方式，这能使得 Form 列数变化后（比如以后又增加了一列组件）这两个按钮始终能保持居中地显示在 Form 最下面。相反，如果采用类似 *addRow(null, okButton, cancelButton)* 的方式，则当列数增大或减少后，按钮将可能不再位于居中的位置，读者可自行实验以观效果。

本节知识点：

Form：Form 继承自 JPanel，因此它是一个单纯的容器，同时它又实现了 LayoutManager 接口，如此便可以自行管理布局而无需额外为它设置布局管理器。Form 对于直接包含的 FormRow 实例，能够采用其表格布局特性来进行布局，对于直接包含的非 FormRow 实例，将表现为类似 SoftBoxLayout 布局的方式。 Form 特有的方法有 addRow，加入一行组件；setVGap，设置行与行之间的间隔；setHGap，设置列与列之间的间隔。

FormRow：它是专门用于辅助 Form 布局的容器，可视为 Form 中的行，一般不用直接创建 FormRow 实例，Form 的 addRow 方法会自动为 Form 创建它的实例，当然，你也可以手动创建 FormRow 实例然后调用 Form.append 方法，效果和 addRow 一样。

2.4 显示日程（JTable 的 MVC 模式）

一开始我们就计划采用表格 JTable 来显示日程列表，上一节我们已经构建了新建日程的界面并通过界面能够得到日程 Task 对象，这一节我们需要把陆续新建的日程列表显示到 JTable 中。

在 AsWing 中，大部分组件都是 MVC 模式的。对于简单的组件，组件内部会创建相应的 Model，我们使用的时候通常不必关心 Model 的存在。比如 JButton，每个 JButton 都有 ButtonModel，大部分时候使用默认的 DefaultButtonModel 即可。但是对于数据比较复杂的组件，由于组件数据通常跟程序数据直接相关，使用者则必须关心 Model 的使用。比如 JList，JTable，JTree 等，每个组件的 Model 都会是一个接口，AsWing 提供一些默认方案的实现：JList 的 VectorListModel，JTable 的 DefaultTableModel 和 PropertyTableModel，JTree 的 DefaultTreeModel，具体到不同的需求，是使用自带的 Model 实现还是自己编写实现，则需使用者自行考量。

2.4.1 日程数据管理和显示（JTable，TableModel）

在我们的日程列表显示需求里，每一行应该是显示一个日程的各个属性，每一列应该对应不同的日程对象。因此，我们选取 PropertyTableModel 最合适， PropertyTableModel 正是管理对象列表，用来把同一对象的不同属性分配到同一行里不同的列来显示。

我们首先创建 TaskManager 类来管理 Task 列表，并且提供装载 Task 列表的 TableModel，编写代码如下：

TaskManager.as

```
package book.scheme{

import org.aswing.VectorListModel;
import org.aswing.table.PropertyTableModel;
import org.aswing.table.TableModel;

/**
 * 日程数据管理器
 */
public class TaskManager{

    private var tableModel:PropertyTableModel;
    private var taskList:VectorListModel;
```

```
    public function TaskManager(){
        taskList = new VectorListModel();
        tableModel = new PropertyTableModel(
            taskList,
            ["名称", "描述", "开始", "时长", "重要度", "状态", "备注"],
            ["name", "description", "startTime", "processTime",
                "importance", "status", "statusComment"],
            [null, null, null, null, null, null, null]
        );
    }

    public function addTask(task:Task):void{
        taskList.append(task);
    }

    public function getTableModel():TableModel{
        return tableModel;
    }
}
}
```

此类的重点在与提供了添加日程的方法 addTask 和提供日程表格数据模型的 getTableModel 方法，内部实现上，采用了我们前面选择的 PropertyTableModel。PropertyTableModel 重点在于构造函数，目前不了解其用法没关系，我们将在稍后对此类进行详细的讲解。

有了数据管理类之后，我们需要在主类中创建此类的实例，然后给添加日程的菜单项和工具栏按钮添加事件监听，在监听函数中调用新建日程控制器创建出日程，然后通过数据管理类添加。

为此，我们需要在主类中重点添加如下两个方法：

```
    private function initControllers():void{
        //当播放器舞台大小时同步显示窗口的大小
        stage.addEventListener(Event.RESIZE, __stageResized);

    tasksContainer.getTable().setModel(taskManager.getTableModel());
        menuContainer.getAddTastMenu().addActionListener(__addTask);
        toolContainer.getAddTaskButton().addActionListener(__addTask);
    }

    private function __addTask(e:Event):void{
        new CreateTaskController(function(task:Task):void{
            if(task){
                taskManager.addTask(task);
            }
        }).show();
```

```
    }
```

（这里省略了部分代码，细节可参见本节源代码）

编译并运行项目，点击添加日程按钮或者菜单项，添加几个日程，之后形成的界面如下：

（图 15）

哈哈，数据终于能够新建并显示出来了。如此简单的几句代码，即构建出了拥有如此复杂数据的表格显示，这全靠 PropertyTableModel 的灵活性。

这里回头看看 TaskManager 中创建 PropertyTableModel 实例部分的代码，构造函数第一个参数传入 taskList，它装载着 Task 对象的序列，第二个参数是一个数组，代表从左到右各列的表头，第三个参数是对应于表头要显示的 Task 数据的属性名（即 Task 对象的属性），第四个参数也是一个数组，用于属性值的转换，这里我们全部传入 null，代表不需要转换。整段构造函数代码的意思是，表格模型的各行数据对象由 taskList 列表提供，每行需要显示**名称、描述、开始、时长、重要度、状态、备注**七个数据，这七个数据分别对应于每个数据对象的 **name、description、startTime、processTime、importance、status、 statusComment** 属性。

尝试再点击添加日程按钮添加一个日程，可以看到表格立刻在最后显示出了这个刚刚添加的数据。但是，我们并没有操作表格 JTable，那么 JTable 是如何获得新的数据并显示出来的呢？这是由于 JTable 的 MVC 设计，当 Model 即 PropertyTableModel 的数据发生变化时（添加、删除或改变某条），Model 会发出一个数据改变的事件，这时候 JTable 的 TableUI 会捕获这个事件，并根据事件中描述的改变来刷新显示界面。而这一切，都是在 JTable 及相关类中自动发生的，使用者不需关心其中的细节。这种架构的优势很明显，使用者关心更少的细节，操作更少的类，获得同样多的能力。

目前我们的表格显示还并不完美，开始时间的显示与我们前面使用的日期格式不符，另外时长与重要度等内容显示尺寸过长，显得空余太多，不合理。对于开始时间的显示，默认

在没有指定 PropertyTranslator 的情况下，PropertyTableModel 会直接返回数据对象的指定属性值，由于 Task.startTime 是 Date 类型，因此在显示为字符串的时候，就成了图 15 中的“Sat Dec 13...”这样的格式，这也是由 Data 对象的 toString 方法决定的。对此，我们可以通过 PropertyTranslator 来进行调整，PropertyTranslator 可以是 PropertyTranslator 对象，也可以是一个 Function 对象，它的作用是把数据原始值转换为要显示出来的值，我们修改构造函数部分代码如下：

```
        tableModel = new PropertyTableModel(
            taskList,
            ["名称", "描述", "开始", "时长", "重要度", "状态", "备注"],
            ["name", "description", "startTime", "processTime",
               "importance", "status", "statusComment"],
            [null, null,
             function(info:*, key:String):*{
               return Utils.dateToString(info[key]);
             }
            , null, null, null, null]
        );
```

当然，也可以这样来实现，更加强类型：

```
        tableModel = new PropertyTableModel(
            taskList,
            ["名称", "描述", "开始", "时长", "重要度", "状态", "备注"],
            ["name", "description", "startTime", "processTime",
               "importance", "status", "statusComment"],
            [null, null,
             function(task:Task, key:String):String{
               return Utils.dateToString(task.startTime);
             }
            , null, null, null, null]
        );
```

调整时长等列的宽度，我们可以通过在主类的 initControllers 方法里添加如下语句调用来实现：

```
    private function initControllers():void{
        //当播放器舞台大小时同步显示窗口的大小
        stage.addEventListener(Event.RESIZE, __stageResized);

    tasksContainer.getTable().setModel(taskManager.getTableModel());
        tasksContainer.getTable().getColumnAt(3).setPreferredWidth(15);
        tasksContainer.getTable().getColumnAt(4).setPreferredWidth(20);
        tasksContainer.getTable().getColumnAt(5).setPreferredWidth(15);

        menuContainer.getAddTastMenu().addActionListener(__addTask);
        toolContainer.getAddTaskButton().addActionListener(__addTask);
    }
```

添加的这三行代码获得指定栏 Column 设置其期望宽度，这里的期望宽度是相对于默认

值 75 的一个相对值，如果设置的值大于 75，那么它将会被分配比其他默认值栏更大的宽度，反之更小。注意这三句代码必须要在给 JTable 设置了 Model 之后再调用，因为拥有 Model 之前，JTable 的栏 Column 还没有与 Model 的栏对应，比如我们这里 TableModel 有 7 栏数据，在设置 Model 之前，JTable 并不知道，因此也就无法设置各栏的期望宽度。

再次编译并运行项目，添加几个日程，之后形成的界面如下：

（图 16）

可以看到图 16 比图 15 更加美观合理。

本节知识点：

JTable：数据表格组件，在一些组件库中，它也被命名为 DataGrid。在 AsWing 中，JTable 由 TableModel 提供数据，JTable 本身实现了 Viewportable 接口，因此可以直接被 JScrollPane 包含，拥有滚动功能。JTable 是一个大型组件，拥有很多操作接口，在后面章节还将陆续讲解。

TableModel：表格模型，它是 JTable 的数据提供者。它通过自身的接口提供表头，各行列数据，并在数据发生更改时发出事件。TableModel 是一个接口，因此任何完整的实现都可以用于向 JTable 提供数据，AsWing 自带的实现有 DefaultTableModel 和 PropertyTableModel。

PropertyTableModel：这是一个 TableModel 的实现。常常用于把一个对象列表中的对象数据，以对象为行，以对象属性为列来显示，它提供默认方式把属性值以直接字符串化显示，

也提供转换器（PropertyTranslator 或 Function）把值转换成期望的字符串格式来显示。

TableColumn：它是 JTable 中的一个列。通常不必手动创建它，当 JTable 创建时，或者数据模型列数改变时，相应的 TableColumn 就会被创建，通过 JTable 的 getColumnAt 或者 getColumn 方法可以得到指定的 TableColumn 对象，常用的 TableColumn 的方法为 setPreferredWidth，设置它相对于默认值 75 的大小，各列不同的期望宽度形成一个比例关系，JTable 根据这个比例关系来分配初始化时各列的宽度。

2.4.2 日程的排序（TableSorter）

上一节我们已经完成了日程列表的显示功能。目前日程的显示顺序是单一地按照日程添加的先后由上而下，除此之外我们还需要按照日期，重要度，状态，甚至是名称来排序，以满足不同的优先查阅需要。

这一节我们将为日程表格添加排序功能，既然要为了不同的需要根据不同的属性来排序，我们便可以为所有表头都添加一个排序功能。表格排序需要用到 TableSorter 类，并且针对不同类型的属性，需要指定不同的比较算法，比如日期需要比较日期的前后按照先后排序，重要度则按照数值排序，名称就只能按字符排序。由此，我们给 TaskManager 类添加如下代码：

```
    public function initTable(table:JTable):void{
        var sorter:TableSorter = new TableSorter(tableModel);
        table.setModel(sorter);
        sorter.setTableHeader(table.getTableHeader());
        tableModel.setColumnClass(2, "Date");
        tableModel.setColumnClass(3, "Number");
        tableModel.setColumnClass(4, "Number");

        sorter.setColumnComparator("Date", __dateCompare);

        table.getColumnAt(3).setPreferredWidth(15);
        table.getColumnAt(4).setPreferredWidth(20);
        table.getColumnAt(5).setPreferredWidth(15);
    }

    private function __dateCompare(o1:String, o2:String):int{
        var date1:Date = Utils.stringToDate(o1);
        var date2:Date = Utils.stringToDate(o2);
        if(date1.getTime() < date2.getTime()){
            return -1;
        }else if(date1.getTime() > date2.getTime()){
            return 1;
        }else{
```

```
            return 0;
        }
    }
```

修改主类的 initControllers 函数为：

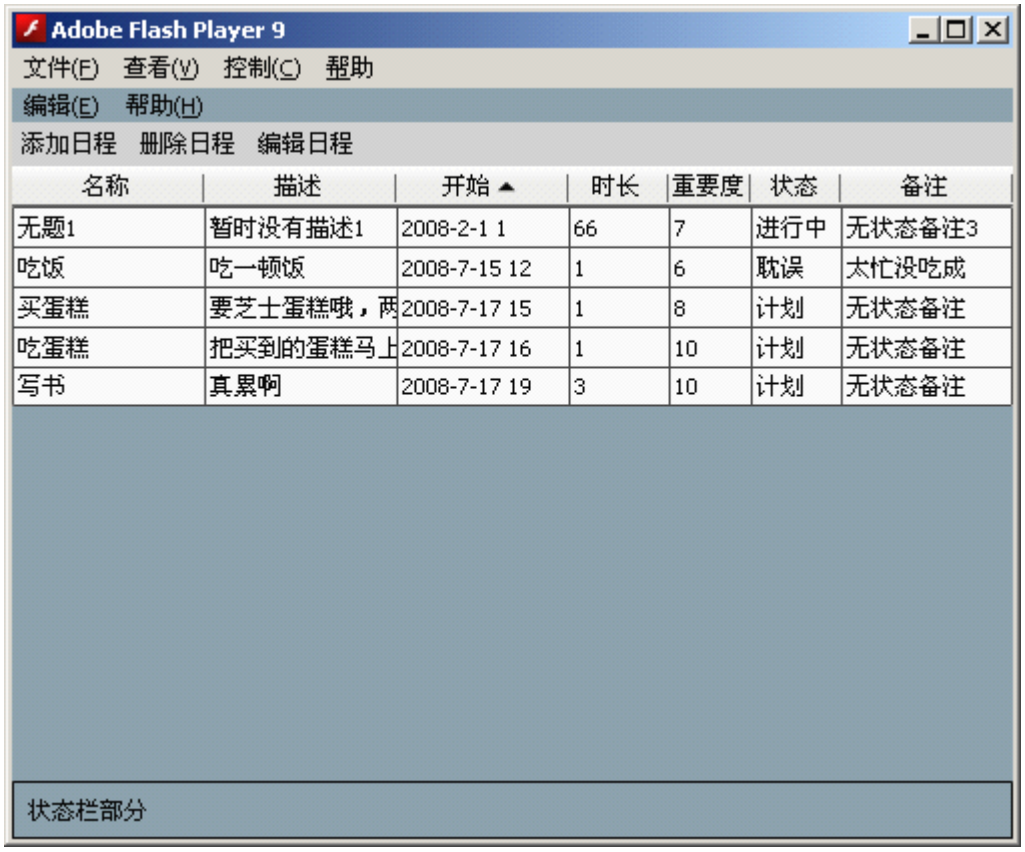
```
    private function initControllers():void{
        //当播放器舞台大小时同步显示窗口的大小
        stage.addEventListener(Event.RESIZE, __stageResized);
        taskManager.initTable(tasksContainer.getTable());

        menuContainer.getAddTastMenu().addActionListener(__addTask);
        toolContainer.getAddTaskButton().addActionListener(__addTask);
    }
```

为了避免主类越来越大，我们把对 JTable 的操作移到 TaskManager 类，因为下面的操作将与 TaskManager 中的 tableModel 直接相关，便于管理。

给 JTable 添加排序能力并不复杂，主要是创建 TableSorter 类包装 tableModel，然后把 JTable 的 tableHeader 传给 TableSorter，这是因为 TableSorter 需要给表头添加鼠标侦听和排序箭头。有了 TableSorter 之后，默认是所有列都拥有排序能力，并且是以字符比较方式来排序的，因此对于数字列，我们需要设置成数值排序，对于日期列，我们需要设置成按日期排序。TableSorter 内置了 *String*（默认）和 *Number* 的排序比较函数，因此对于第三列（时长）和第四列（重要度），我们只需要把 columnClass 设置成 *Number* 即可，而对于日期，我们则需要自己实现比较函数，因此我们定义了__dateCompare 函数，并且给 *Date* 类型设置成需要用它做比较函数。

注意，这里我们遇到了 columnClass（列类型）概念，在 TableModel 接口中定义了 *getColumnClass(columnIndex:int):String* 方法，它返回指定列的类型，列类型主要用来分辨一个列的类型并以此来给列安排排序比较函数和单元格渲染方式(下一节我们将会讲解自定义单元格渲染)，这里我们给 2，3，4 列设置了列类型，因此 2，3，4 就能根据列的类型来自动选择排序比较函数，对于未设置列类型的列，将采用默认的列类型——*Object* 和默认的比较函数——字符比较。

编译并运行程序，添加一些日程，点击“开始”表头，我们将得到如下界面：



（图 17）

这是按照开始日期来排序得到的表格，读者可以尝试点击其它表头来获得不同的排序方式，如果我们希望某一些列不能排序，可以通过 *TableSorter.setColumnSortable(column:int, sortable:Boolean):void* 来指定。如果我们希望通过程序调用的方式来改变排列方式，可以调用 *TableSorter.setSortingStatus(column:int, status:int):void* 来实现，读者可以自行尝试。

本节知识点：

TableSorter：表格排序器。它是 TableModel 接口的一个实现，通过包装一个现有的 TableModel 来实现排序。实现原理为：以选择排序方式为基准，变换原始 TableModel 类的行的值，以达到变换行顺序的目的。排序所采用的值比较函数，可以通过给指定列设定列类型，然后给指定列类型设定比较函数来达到。默认情况下，所有列的类型都是 *Object* 类型，TableSorter 自带 *String* 和 *Number* 类型的比较函数，对于没有指定比较函数的列，默认将采用 *String* 类型的比较函数。

columClass：列类型。它属于 TableModel 的内容，主要目的是为了让不同类型的列能使用格子的排序比较函数和单元格渲染方式。

2.4.3 更改单元格颜色 （自定义 TableCell）

为了突出一些特定状态的日程，通常需要给它们赋予不同的字体颜色或者背景色。这一节我们尝试实现这些功能。

假设，我们要突出临近的日程，并且把不同状态的日程明显区分开，我们可以给临近 24 小时需要开始的日程开始时间格用橙色背景，把进行中的日程状态格用绿色字体，已完成的用蓝色字体，中断或耽误的用红色字体，其他的则保持原有格式不变。

要实现这样的功能，我们需要自定义单元格工厂给指定类型的列生产单元格，根据此需求，我们编写具备自定义颜色功能的表格单元格类：

ColorTableCell.as

```
package book.scheme{

import org.aswing.ASColor;
import org.aswing.JTable;
import org.aswing.table.PoorTextCell;

public class ColorTableCell extends PoorTextCell{

    private var fgColor:ASColor;
    private var bgColor:ASColor;

    public function ColorTableCell(){
        super();
    }

    override public function setCellValue(value:*):void{
        super.setCellValue(value);
        //对于颜色单元格的值类型，保存颜色设置
        var ccv:ColorCellValue = value as ColorCellValue;
        if(ccv){
            fgColor = ccv.fgColor;
            bgColor = ccv.bgColor;
            setText(ccv.text);
        }else{
            fgColor = null;
            bgColor = null;
        }
    }

    override public function setTableCellStatus(table:JTable,
selected:Boolean, row:int, column:int):void{
        super.setTableCellStatus(table, selected, row, column);
        //设置颜色
        if(fgColor != null){
            setForeground(fgColor);
        }
```

```
        if(bgColor != null){
            setBackground(bgColor);
        }
    }
}
}
```

ColorCellValue.as

```
package book.scheme{

import org.aswing.ASColor;

public class ColorCellValue{

    public var fgColor:ASColor;
    public var bgColor:ASColor;
    public var text:String;
}
}
```

表格的单元格必须实现自接口 TableCell，由于我们只需要给单元格添加自定义颜色功能，因此我们继承自 AsWing 自带的 PoorTextCell 类，它是提供文本显示能力的表格单元格。我们覆盖了 PoorTextCell 类的两个方法，分别为设置值和设置状态的方法，在 setCellValue 中，记录下单元格值中附带的颜色属性，然后在 setTableCellStatus 中把保存下来的颜色应用起来。注意这两个覆盖的方法都调用了父类的方法 super.XXXX()以保持原本父类的功能不被破坏。感兴趣的读者可以阅读 PoorTextCell 类的源代码，观察它的这两个方法是如何实现的，为什么我们在覆盖方法内还必须调用它们。

由于 ColorTableCell 只针对 ColorCellValue 类型的值拥有颜色定义能力，所以我们必须修改 TaskManager 内 tableModel 的各相应 PropertyTranslator，让它们能返回此类型的值。并且，还需要通过 JTable.setDefaultCellFactory 给开始时间和状态列设置能产生 ColorTableCell 的单元格工厂。我们修改 TaskManager 类最终如下：

```
package book.scheme{

import org.aswing.ASColor;
import org.aswing.JTable;
import org.aswing.VectorListModel;
import org.aswing.table.GeneralTableCellFactory;
import org.aswing.table.PropertyTableModel;
import org.aswing.table.sorter.TableSorter;

/**
 * 日程数据管理器
 */
public class TaskManager{
```

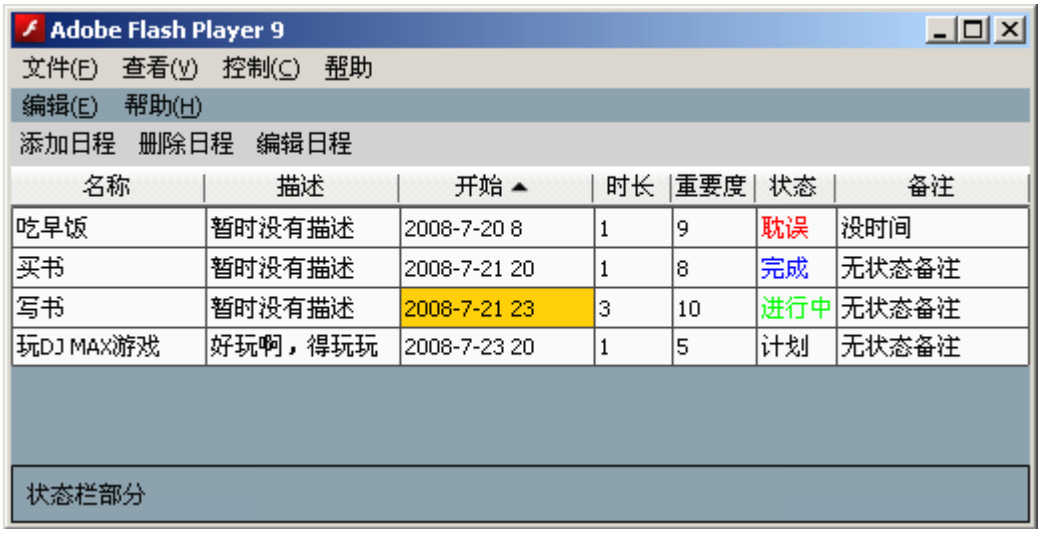
```
private var tableModel:PropertyTableModel;
private var taskList:VectorListModel;

public function TaskManager(){
    taskList = new VectorListModel();
    tableModel = new PropertyTableModel(
        taskList,
        ["名称", "描述", "开始", "时长", "重要度", "状态", "备注"],
        ["name", "description", "startTime", "processTime",
            "importance", "status", "statusComment"],
        [null, null,
         __dateTranslator
        , null, null,
        __statusTranslator, null]
    );
}

private function __dateTranslator(task:Task, key:String):*{
    var value:ColorCellValue = new ColorCellValue();
    value.text = Utils.dateToString(task.startTime);
    var now:Date = new Date();
    var dis:Number = (task.startTime.getTime()
        - now.getTime())/1000/60/60;
    if(dis > 0 && dis < 24){
        value.bgColor = ASColor.ORANGE;
    }
    return value;
}

private function __statusTranslator(task:Task, key:String):*{
    var value:ColorCellValue = new ColorCellValue();
    value.text = task.status;
    if(task.status == TaskStatus.FINISHED){
        value.fgColor = ASColor.BLUE;
    }else if(task.status == TaskStatus.INTERRUPTED
        || task.status == TaskStatus.MISS){
        value.fgColor = ASColor.RED;
    }else if(task.status == TaskStatus.PROCESSING){
        value.fgColor = ASColor.GREEN;
    }
    return value;
}
```

```
    public function initTable(table:JTable):void{
        var sorter:TableSorter = new TableSorter(tableModel);
        table.setModel(sorter);
        sorter.setTableHeader(table.getTableHeader());
        sorter.setColumnSortable(1, false);//可以使指定列不可排序
        sorter.setSortingStatus(3, TableSorter.ASCENDING);//默认排序
        tableModel.setColumnClass(2, "Date");
        tableModel.setColumnClass(3, "Number");
        tableModel.setColumnClass(4, "Number");

        sorter.setColumnComparator("Date", __dateCompare);

        table.getColumnAt(3).setPreferredWidth(15);
        table.getColumnAt(4).setPreferredWidth(20);
        table.getColumnAt(5).setPreferredWidth(15);

        tableModel.setColumnClass(5, "Status");
        //设置指定列类型的单元格工厂
        table.setDefaultCellFactory("Status",
            new GeneralTableCellFactory(ColorTableCell));
        table.setDefaultCellFactory("Date",
            new GeneralTableCellFactory(ColorTableCell));
    }

    private function __dateCompare(o1:ColorCellValue,
        o2:ColorCellValue):int{
        var date1:Date = Utils.stringToDate(o1.text);
        var date2:Date = Utils.stringToDate(o2.text);
        if(date1.getTime() < date2.getTime()){
            return -1;
        }else if(date1.getTime() > date2.getTime()){
            return 1;
        }else{
            return 0;
        }
    }

    public function addTask(task:Task):void{
        taskList.append(task);
    }
}
}
```

我们通过__dateTranslator 和__statusTranslator 两个转换器，分别生成开始时间列和状态列的 ColorCellValue 对象，并且给状态列设置类型为 Status，最后给 Date 和 Status

列设置单元格工厂为 *new GeneralTableCellFactory(ColorTableCell)*，GeneralTableCellFactory 会为每个对应的单元格创建 ColorTableCell 实例。注意由于开始时间列单元的值类型由原来的 String 变成了 ColorCellValue，所以__dateCompare 方法也做了相应的调整。

编译并运行项目，添加一些日程，我们将得到如下界面：



（图 18）

由此可见，达到了我们想要的效果。

本节知识点：

TableCell：表格单元格接口。表格界面是由表头+单元格+分界线组成，单元格负责描绘数据单元。默认情况下，表格采用 PoorTextCell 单元格，负责把单元格值以字符串文本的形式描绘出来。如果开发者需要改变描绘方式，比如用特殊的颜色，或者用图形，或者用其他组件如选择框，下拉框，都可以通过实现自己的 TableCell 来达到。TableCell 最主要的两个接口是 *setCellValue(value:*) : void* 和 *setTableCellStatus(table : JTable, selected : Boolean, row : int, column : int) : void*，前者意为给单元格设置要描绘的值，后者意为设置单元格的状态，通常，在前者中为单元格组件设置要描绘的内容，在后者为单元格组件设置字体和颜色。另外，*getCellComponent() : Component* 接口则是返回要描绘单元格的组件。读者可以参考 AsWing 自带的 PoorTextCell 和 DefaultTableCell 两个类来研究单元格的实现方法。理论上，任何组件都可以作为单元格组件，因此，AsWing 的表格拥有极其灵活的表现方式，你可以在单元格里放置一个拥有大量组件的容器，或者一个树，甚至是表格——形成表格中的表格。

TableCellFactory：单元格工厂。产生单元格实例的工厂，通常使用 GeneralTableCellFactory 即可，单元格工厂只是用来创建单元格实例的类，因此并不复杂，读者可以参考 GeneralTableCellFactory 类的源代码理解其意义。

columClass：列类型。上一节和本节都用到了它，以后在设置单元格编辑器时，还会用到它。

2.5 管理日程

我们已经能够添加并比较美观的显示出日程了，这一大节，我们将专注于管理现有的日程，包括显示细节，编辑或删除已有日程。

2.5.1 显示细节（JTable 选择事件）

由于我们采用的表格单元都是单行文字，因此对于描述和备注这两栏信息，可能会存在显示面积不够的情况，但是又不值得仅仅因为这两项而把整个表格的行都变成多行，这样会限制整体显示日程的数量。为此，我们可以把描述和备注内容在另一个地方完整显示出来，并且只显示选中日程的内容，这样既保证了整体日程数量的显示能力，又能看到完整的描述和备注信息。

我们可以在状态栏的位置显示这些内容，先编写显示界面类如下：

TaskDetailsPane.as

```
package book.scheme{

import org.aswing.BorderLayout;
import org.aswing.BoxLayout;
import org.aswing.JLabel;
import org.aswing.JPanel;
import org.aswing.JScrollPane;
import org.aswing.JTextArea;

public class TaskDetailsPane extends JPanel{

    private var descriptionText:JTextArea;
    private var statusComment:JTextArea;

    public function TaskDetailsPane(){
        super(new BoxLayout());
        descriptionText = new JTextArea("", 4);
        descriptionText.setWordWrap(true);
        descriptionText.setEditable(false);
        statusComment = new JTextArea("", 4);
        statusComment.setWordWrap(true);
        statusComment.setEditable(false);
```

```
        var destPane:JPanel = new JPanel(new BorderLayout());
        var destLabel:JLabel = new JLabel("描述：");
        destPane.append(destLabel, BorderLayout.WEST);
        destPane.append(new JScrollPane(descriptionText),
            BorderLayout.CENTER);

        var commentPane:JPanel = new JPanel(new BorderLayout());
        var commentLabel:JLabel = new JLabel("备注：");
        commentPane.append(commentLabel, BorderLayout.WEST);
        commentPane.append(new JScrollPane(statusComment),
            BorderLayout.CENTER);

        appendAll(destPane, commentPane);
    }

    public function getDescriptionText():JTextArea{
        return descriptionText;
    }

    public function getStatusComment():JTextArea{
        return statusComment;
    }
}
}
```

我们把这个界面平分为左右两个部分，左边显示描述，右边显示备注，并且描述和备注框都设置为自动换行和不可编辑，因为这里只是显示用，开启编辑能力反而会影响用户感受。在布局上，采用 BoxLayout 平分左右，然后各部分又用 BorderLayout 来装载标签和文本框，文本框用一个 JScrollPane 包装以获得滚动条能力。这样的布局策略，使得两个文本框均能平均获得最大的显示宽度，显示高度设死了为 4 行文本高度。

由于需要获知用户对表格的选择行为，以便在选择变化时更新显示内容，所以我们需要监听表格的选择事件，我们在表格数据的管理类 TaskManager 中做这一监听，主要给 TaskManager 添加如下代码：

```
    /**
     * 设置当选中的日程改变时的处理函数，调用格式为 handler(task:Task)
     */
    public function setTaskChangeHandler(handler:Function):void{
        taskChangeHandler = handler;
    }

    public function initTable(table:JTable):void{
        this.table = table;
```

```
        sorter = new TableSorter(tableModel);

        ......省略部分代码

        table.addEventListener(SelectionEvent.ROW_SELECTION_CHANGED,
            __rowSelectionChanged);
    }

    private function __rowSelectionChanged(e:SelectionEvent):void{
        var row:int = table.getSelectedRow();
        var task:Task = null;
        if(row >= 0){//>=0代表有选中的行
            task = taskList.getElementAt(sorter.modelIndex(row));
        }
        //调用日程改变时的处理函数
        taskChangeHandler(task);
    }
```

为了节约篇幅，这里没有列出完整的类代码。添加的代码部分，其主要作用是让 TaskManager 监听行选择的变化事件，当行选择变化时，获取当前选中的行，由于表格有排序器 TableSorter，所以直接获取的行，要通过 TableSorder 的 modelIndex 方法转换一下才是真正数据模型中的行号，由此行号得到选中的 Task 对象，然后触发日程改变处理函数。

这里添加的日程改变处理函数功能，目的是为了让 TaskManager 外界能够获得这一行为并得到数据，由此我们可以编写 TaskDetailsController 类来根据这个设定为 TaskDetailsPane 填充数据，代码如下：

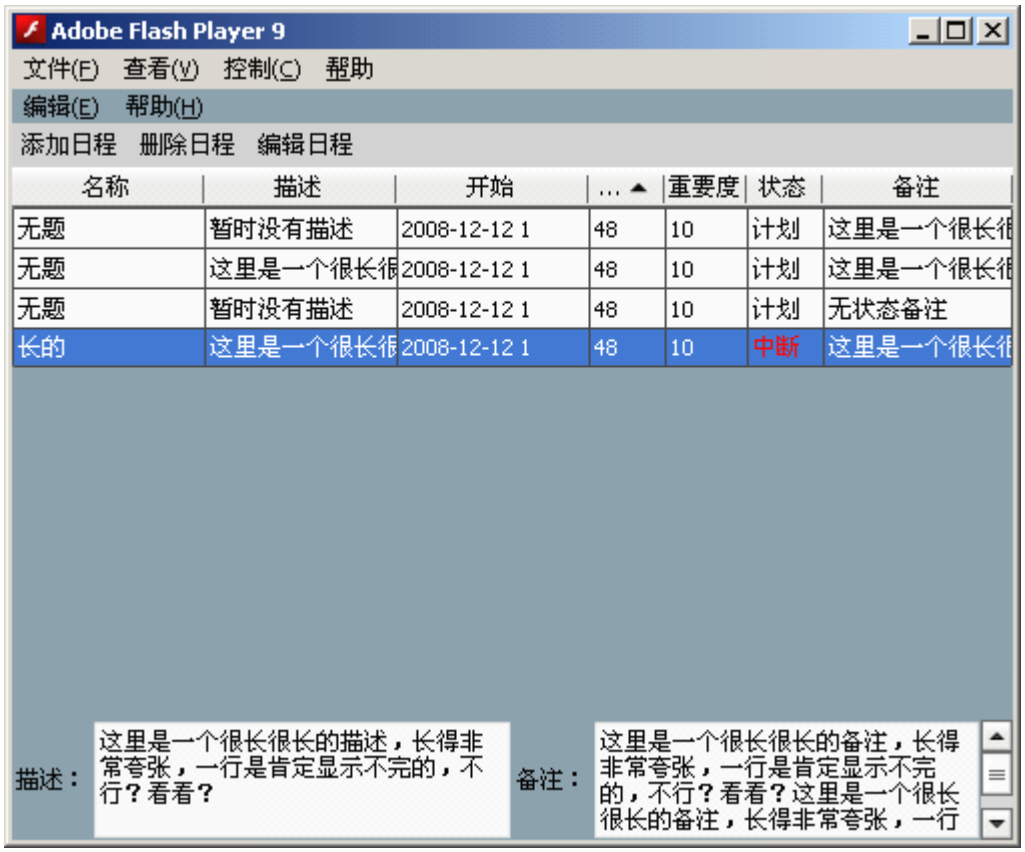
TaskDetailsController.as

```
package book.scheme{

public class TaskDetailsController{

    private var pane:TaskDetailsPane;

    public function TaskDetailsController(){
        pane = new TaskDetailsPane();
    }

    public function initControler(manager:TaskManager):void{
        manager.setTaskChangeHandler(__taskChanged);
    }

    private function __taskChanged(task:Task):void{
        if(task){
            pane.getDescriptionText().setText(task.description);
            pane.getStatusComment().setText(task.statusComment);
```

```
        }else{
            pane.getDescriptionText().setText("");
            pane.getStatusComment().setText("");
        }
    }

    public function getPane():TaskDetailsPane{
        return pane;
    }

}
}
```

此控制器类非常简单，给TaskManager 设置一个日程改变处理函数，然后当日程改变时，用处理函数被传入的当前日程，给 TaskDetailsPane 的两个文本框设置相应内容，即达到显示当前日程细节的能力。

要让这一套机制运作起来，我们还需要更改主类，把主类中原来的状态栏用现在的TaskDetailsController 及其含有的 TaskDetailsPane 替代，并且在 `initControllers` 方法中初始化 TaskDetailsController，编译并运行程序，添加一些日程，选中一个拥有较多描述和备注文字的日程（在新建一个含有较多描述和备注文字的日程的时候，我们发现新建日程面板的这两个输入框不会自动换行，使用不是很方便，顺带把这两个输入框也通过 *setWordWrap(true)*改成了自动换行），将看到如下界面：



（图 19）

虽然细节显示这部分界面好像不是很美观，但是我们能看到详细的文字内容了，并且当文字比较多时还自动出现了滚动条，功能算是完备了。

本节知识点：

SelectionEvent：选择事件。表格的行，列（以及列表 List）共用的选择事件类。SelectionEvent.ROW_SELECTION_CHANGED 事件代表行的选择改变时发出的事件，相应的 SelectionEvent.COLUMN_SELECTION_CHANGED 则代表列的选择改变时发出的事件，默认情况下，表格 JTable 是关闭了行选择功能（如果需要，可以通过 *JTable.setColumnSelectionAllowed(true)* 来开启它）。由于我们的表格设计为每行显示一个日程，因此当行选择改变时，我们就能通过行号来获取到选中的日程。注意，由于我们使用了 TableSorter 来进行排序，因此显示出来的行的顺序可能和数据模型中顺序不同，因此需要通过 *TableSorter.modelIndex* 方法来进行转换才能得到实际数据模型中的对应的行。

处理函数：在 TaskManager 中，我们实现了一个日程改变的处理函数的逻辑，这是为了封装表格事件与模型的关联细节，对外只暴露出一个处理函数，方便外部简单获得需要的数据，从TaskDetailsController 的简洁即可看出这种方式的便利性。但是这种方式也把处理函数局限到只能有一个的境地，如果还有其他控制器需要处理日程选择的改变事件，那么此方法必须改进，可以通过维护一个处理器列表，把设置处理器改为添加和移除处理器的方式，允许多个处理器同时存在，来达到目的。当然，也可以更传统地让 TaskManager 继承或含有 EventDispatcher，用事件的方式来实现。

2.5.2 修改日程（复用 CreateTaskPane）

如果一个日程临时有变动，或者事后需要改写状态备注，那么我们则需要有修改日程的功能。如果可以修改日程的任意一个属性，那么这个修改界面应该提供和新建日程界面相同数量的组件，实际上，我们可以共用新建日程面板，通过不同的控制器，来实现不同的功能。

我们编写与 CreateTaskController 极度相似的类 EditTaskController 如下：

EditTaskController.as

```
package book.scheme{

import flash.events.Event;

import org.aswing.JFrame;
import org.aswing.JOptionPane;

/**
 * 编辑日程控制器
 */
```

```
public class EditTaskController{

    private var handler:Function;
    private var dialog:JFrame;
    private var pane:CreateTaskPane2;
    private var task:Task;

    /**
     * 构造编辑日程控制器
     * @param handler 编辑结果处理器，格式handler(task:Task)
     *      如果task为null代表取消了编辑
     */
    public function EditTaskController(task:Task,
        handler:Function){

        this.task = task;
        this.handler = handler;
        pane = new CreateTaskPane2();
        pane.getStatusCombo().setListData([
            TaskStatus.PLAN,
            TaskStatus.PROCESSING,
            TaskStatus.CANCEL,
            TaskStatus.FINISHED,
            TaskStatus.INTERRUPTED,
            TaskStatus.MISS
        ]);

        pane.getStatusCombo().setSelectedItem(task.status);
        pane.getNameText().setText(task.name);
        pane.getDescriptionText().setText(task.description);
        pane.getStartTimeText().setText(
            Utils.dateToString(task.startTime));
        pane.getProcessTimeText().setText(task.processTime+"");
        pane.getImportanceText().setText(task.importance+"");
        pane.getStatusComment().setText(task.statusComment);

        dialog = new JFrame(null, "编辑日程", true);
        dialog.setContentPane(pane);

        pane.getOkButton().addActionListener(__ok);
        pane.getCancelButton().addActionListener(__cancel);
    }

    private function __ok(e:Event):void{
```

```
        var startTime:Date = null;
        try{
            startTime = Utils.stringToDate(
                pane.getStartTimeText().getText());
        }catch(error:Error){
            JOptionPane.showMessageDialog("提示", error.message);
            return;
        }
        task.startTime = startTime;
        task.description = pane.getDescriptionText().getText();
        task.importance = Math.min(10,
            int(pane.getImportanceText().getText()));
        task.name = pane.getNameText().getText();
        task.processTime = int(pane.getProcessTimeText().getText());
        task.status = pane.getStatusCombo().getSelectedItem();
        task.statusComment = pane.getStatusComment().getText();
        dialog.dispose();
        handler(task);
    }

    private function __cancel(e:Event):void{
        dialog.dispose();
        handler(null);
    }

    public function show():void{
        dialog.pack();
        dialog.show();
    }
}
}
```

此类与 CreateTaskController 类大部分代码都一样，不过实现目的不一样，此类是通过用户输入的数据修改现有 Task 对象，而 CreateTaskController 是通过用户输入的数据创建一个新的 Task 对象。

学过重构方法的读者可以已经嗅到了这里的坏味道，两个类拥有太多相同的代码，我们应该给这两个类提取出一个共同的基类，把相同的逻辑部分提取出来，简化代码，便于扩展和维护。本章重点不在讲解此方面的技术，因此把这个工作留给读者（提示，可以通过重点提取出一个 *checkInputValid():Boolean* 和 *fillDate(task:Task):void* 这样的两个方法给基类）。

由于需要给编辑控制器提供 Task 对象，因此我们还必须有办法获得当前选中的 Task，当点击编辑日程按钮或者菜单项时，通过选中的 Task 对象来创建编辑控制器。由此，我们需要给主类添加如下代码：

```
    private function initControllers():void{
        ......//省略以往代码
        toolContainer.getAddTaskButton().addActionListener(__addTask);
        menuContainer.getEditTastMenu().addActionListener(__editTask);
    toolContainer.getEditTaskButton().addActionListener(__editTask);
        //模拟一次选中日程变化，初始化相关组件状态
        __taskSelectionChanged(null);
    }

    private function __taskSelectionChanged(task:Task):void{
        var selection:Boolean = (task != null);
        menuContainer.getEditTastMenu().setEnabled(selection);
        toolContainer.getEditTaskButton().setEnabled(selection);
    }

    private function __editTask(e:Event):void{
        var task:Task = taskManager.getSelectedTask();
        if(task){
            new EditTaskController(task,
                function(editedTask:Task):void{
                if(editedTask){
                    taskManager.notifyTaskChanged(editedTask);
                }
            }).show();
        }
    }
```

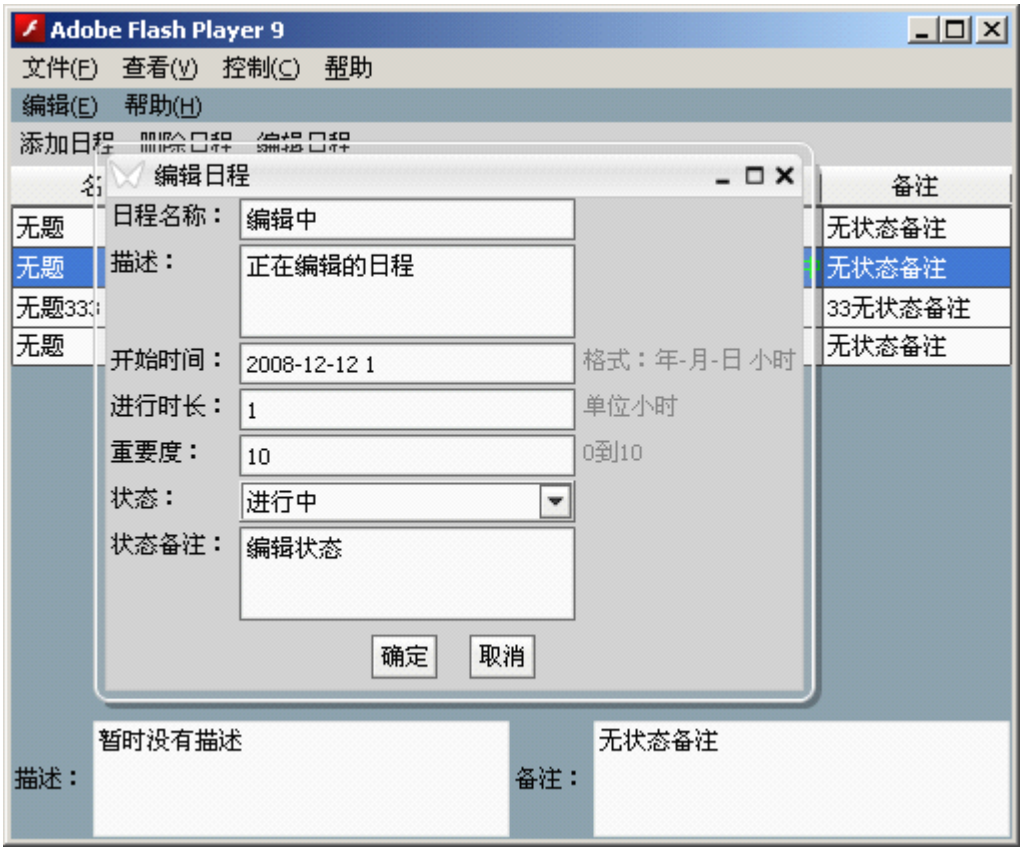
通过给编辑菜单项和编辑按钮添加动作事件来调用编辑日程函数，函数体内首先通过 TaskManager 拿到当前选中的 Task，如果存在，则创建编辑控制器进行编辑，原理和新建日程类似。

注意，为了使得只有在某 Task 被选中时，才使编辑按钮和菜单项有效，我们也设置了选中日程改变函数__taskSelectionChanged。在这之前，其实我们对 TaskManager 也进行了改造，上一节末我们提到，当还有别的地方需要对日程选择改变进行处理时，TaskManager 需要维护一个处理器列表，因此我们把原来的单个处理器改为了处理器数组，setTaskChangeHandler 函数改为了 addTaskChangeHandler，并且增加了 getSelectedTask 函数得到当前选中的日程对象，另外还增加了 notifyTaskChanged 函数，它的作用是当某一个日程的内容被更改（编辑）了时，触发相关事件以通知相关界面进行更新。变动相关代码为：

```
    /**
     * 设置当选中的日程改变时的处理函数，调用格式为 handler(task:Task)
     */
    public function addTaskChangeHandler(handler:Function):void{
        taskChangeHandlers.push(handler);
    }
```

```
    /**
     * 通知Model某个Task内容改变了
     */
    public function notifyTaskChanged(task:Task):void{
        taskList.valueChanged(task);
        //如果选中的task内容改变了，通过此调用选中的日程改变时的处理函数
        if(task == getSelectedTask()){
            __rowSelectionChanged(null);
        }
    }

    /**
     * 返回当前选中的 Task
     */
    public function getSelectedTask():Task{
        var row:int = table.getSelectedRow();
        var task:Task = null;
        if(row >= 0){//>=0代表有选中的行
            task = taskList.getElementAt(sorter.modelIndex(row));
        }
        return task;
    }

    private function __rowSelectionChanged(e:SelectionEvent):void{
        var task:Task = getSelectedTask();
        //调用日程改变时的处理函数
        for each(var handler:Function in taskChangeHandlers){
            handler(task);
        }
    }
```

编译并运行现在的项目，添加一些日程，选中其中一个，再点击编辑日程按钮，我们将看到类似如下的界面：



（图 20）

当点击确定之后，我们将看到表格和细节界面，都将得到更新。

本节知识点：

界面更新：对于标准的 MVC 模式的组件，只需要通知 Model 某些数据改变了，Model 会自己广播事件给 Controller，Controller 则会根据事件内容对 View 做内容更改，因此界面更新工作非常简单。比如本节对 `TaskManager` 添加的 `notifyTaskChanged` 函数中的第一句，它通知列表模型某个 Task 更改了，然后表格自动就会显示出更改后的新内容，这是因为列表模型通知了包含它的表格模型（PropertyTableModel），表格模型会发出事件告诉表格的控制器，然后表格控制器则会对表格界面进行更新，这一过程已经在 AsWing 的 JTable 及相关类中实现好了，用户使用起来不用关心其中的细节。而对于我们自己编写的日程细节显示部分，这并不是一个 MVC 模式的实现，由于其功能简单，并且不需要复用，因此我们也没必要大费周折的把它用 MVC 来实现一次，我们只需要在 `notifyTaskChanged` 函数中用了一个 `if` 判断，然后再借用 `__rowSelectionChanged` 现有的逻辑，即实现了更新它的界面的能力。

组件有效性设置：为了提升用户感受，我们通常只让能用的组件处于可点击状态，比如本节对编辑日程按钮及菜单的设置，当没有选中任何日程时，它们不应该处于有效状态，这会误导用户，而当有日程被选中时，它们则应该立刻变为有效状态，这些，一般可以通过监听选择更改事件来进行处理。本节通过改进上一节自制的日程改变处理函数的功能，达到了这个目的。当然，像上一节提到的一样，你也可以用更通用的 Event 方式来实现，同样可以

达到这样的目的。

2.5.3 删除日程（JOptionPane）

三个日程功能按钮和菜单项，现在就剩下删除日程一项了。这一节我想我也不用多讲了，按照前面两节的模式，删除日程实现起来易如反掌，重点也就是调用 VectorListModel 的 remove 方法移除选中的日程。

首先给 TaskManager 添加 removeTask 方法，代码如下：

```
public function removeTask(task:Task):void{
    taskList.remove(task);
}
```

这里留意一下，前面章节我们讲到过通过修改 Model，界面会自动更新，这里把从列表中移除一个日程，同样界面会自动更新。

然后按照上一节编辑选中日程的方式，给删除日程按钮和菜单项添加事件，事件处理函数为：

```
private function __removeTask(e:Event):void{
    var task:Task = taskManager.getSelectedTask();
    if(task == null){
        return;
    }
    var jop:JOptionPane = JOptionPane.showMessageDialog(
        "提示",
        "确认删除日程？",
        function(result:int):void{
            if(result == JOptionPane.YES){
                taskManager.removeTask(task);
            }
        },
        mainWindow, true, null,
        JOptionPane.YES|JOptionPane.NO);
    jop.getFrame().setDefaultButton(jop.getYesButton());
}
```

这段代码是本节最主要的代码了，首先它获取到当前选中的日程，如果存在，那么弹出确认删除的对话框，如果用户点击了 Yes，那么就移除选中的日程，否则不做任何处理。

本节知识点：

JOptionPane：在本章前面的小节里，我们使用过此类并简单介绍过它，这一节我们来详细介绍这个组件。JOptionPane 继承自 JPanel，可以直接创建其实例放置于其他容器中。通常，普通的使用， JOptionPane 的两个静态方法即足够。第一个方法：

```
public static function showMessageDialog(title:String,
    msg:String, finishHandler:Function=null,
    parentComponent:Component=null,
    modal:Boolean=true,
    icon:Icon=null,
    buttons:int=OK):JOptionPane{
```

它的作用是弹出一个含有标题，提示文字，图标，按钮，可模态，并且可传入一个事件处理函数的对话框。各参数的意义分别为：

- **title**：对话框的标题，即弹出的 JFrame 的标题。
- **msg**：对话框主题中显示的文本内容。
- **finishHandler**：当用户点击按钮关闭对话框时调用的处理函数，它的形式为 *finishHandler(result:int):void*，result 的值为用户点击的按钮对应的值，它可能是以下几种之一：

```
public static const OK:int = 1; //00001
public static const CANCEL:int = 2; //00010
public static const YES:int = 4; //00100
public static const NO:int = 8; //01000
public static const CLOSE:int = 16; //10000
```

分别对应于 OK，Cancel，Yes，No，Close 按钮（CLOSE 一项同样作用于窗口的关闭按钮）。

- **parentComponent**：对话框的父组件。指定它，可以让对话框以此父组件的顶级容器的子窗口的形式弹出，让对话框保持始终在父组件之上层显示。此参数可省略为 null，代表没有父组件。
- **modal**：是否为模态的。模态的对话框在关闭之前，用户不能激活模态对话框下层的其他窗口内的组件，使得用户必须完成对话框内的操作，才能继续操作其他组件。
- **icon**：指定对话框主题内要显示的图标。
- **buttons**：指定对话框提供的选择按钮。可以采用“|”运算符指定多个按钮，比如本节的 *JOptionPane.YES|JOptionPane.NO* 指定显示 Yes 和 No 两个按钮。

第二个方法，弹出一个输入对话框，形式如下：

```
public static function showInputDialog(title:String,
    msg:String,
    finishHandler:Function=null,
    defaultValue:String="",
    parentComponent:Component=null,
    modal:Boolean=true,
    icon:Icon=null):JOptionPane{
```

大部分参数与 showMessageDialog 中的参数意义相同，不同的是：

- **finishHandler**：当用户点击按钮关闭对话框时调用的处理函数，它的形式为 *finishHandler(result:String):void*，其中 result 的值为用户在文本输入框输入的字符串，如果它的值为 null，代表用户取消了输入（点击了取消或者关闭按钮）。
- **defaultValue**：对话框弹出时，文本输入框中默认的字符串。

窗口默认按钮：每个 JRootPane（JFrame/JWindow/JPopup 的父类），都可以通过 *setDefaultButton* 方法设置一个默认按钮，即当用户点击键盘回车按键时，触发的按钮。

这对于弹出对话框来说，非常有用，例如本节的这一语句：*jop.getFrame().setDefaultButton(jop.getYesButton());*它给弹出对话框设置 Yes 按钮为默认按钮，那么当用户点击回车时，则相当于点击了 Yes 按钮。

2.5.4 快速修改时长和状态（CellEditor）

第二小节我们已经拥有了日程编辑器，但是对于某一些个别的常常变动的属性，每次改变都要打开整个编辑器，显得过于麻烦，如果能够直接在表格中编辑这些属性，那将是最好的用户感受了，比如一个日程的时长，需要将原定的 44 改为 22，我们希望可以直接在表格单元格中输入“22”后点击确定来实现，而无需专门打开日程编辑窗口来进行修改。

经过分析，我们觉得时长和状态两个属性常常存在变动，而且内容又都很简短，因此非常适合直接在表格单元格中编辑。这需要通过设置 CellEditor 来实现，先看看实现代码，在 TaskManager 类的 initTable 方法中加入如下代码：

```
        //设置单元格编辑器
        tableModel.setColumnEditable(3, true);
        tableModel.setColumnEditable(5, true);

        var timeEditor:DefaultNumberTextFieldCellEditor =
            new DefaultNumberTextFieldCellEditor();
        var statusEditor:DefaultComboBoxCellEditor =
            new DefaultComboBoxCellEditor();
        statusEditor.getComboBox().setListData([
            TaskStatus.PLAN,
            TaskStatus.PROCESSING,
            TaskStatus.CANCEL,
            TaskStatus.FINISHED,
            TaskStatus.INTERRUPTED,
            TaskStatus.MISS
        ]);
        table.getColumnAt(3).setCellEditor(timeEditor);
        table.getColumnAt(5).setCellEditor(statusEditor);
```

首先，给 Model 设置可编辑的列为第三和第五列，对应于时长和状态。然后创建时长编辑器和状态编辑器，分别为一个数字编辑器和一个下拉列表编辑器，然后把这两个编辑器设置给对应的第三和第五列。

编译并运行新的程序，添加一个日程，点击状态单元格，会出现一个下拉列表，界面如下：

（图 21）

可见这种编辑日程状态的方法，比通过打开编辑器来编辑，要方便得多。

本节知识点：

CellEditor：单元格编辑器，它是一个接口，任何完整实现了此接口的类，都是一个单元格编辑器，而不同的组件通常还有自己专属的单元格编辑器子接口，比如表格的单元格编辑器是需要完整实现 TableCellEditor 的（当然它也是 CellEditor 的子类，对应，树的单元格编辑器接口是 TreeCellEditor），但是就目前版本的 AsWing 实现中，TableCellEditor 和 TreeCellEditor 都没有附加的方法，因此它们的各自的实现，是可以轻松共用的。本节中使用了 AsWing 自带的 DefaultNumberTextFieldCellEditor 来编辑时长，是因为时长是数字类型；用 DefaultComboBoxCellEditor 来编辑状态，并填入状态数组，是因为状态是几个字符串值的枚举。AsWing 除了自带这两个编辑器外，还有一个用于编辑字符串的 DefaultTextFieldCellEditor。对于一个特定类型属性的编辑，一定要配上对应的编辑器，否则编辑器返回的值类型不同，程序则会出错，读者可以试着把本节的 3，5 列编辑器对调，然后运行程序试试看是什么后果。

对于无法采用 AsWing 自带编辑器来编辑的属性，开发者必须自己实现对应的编辑器，这里举日程开始时间属性为例，虽然表现形式是一个字符串，但是实际的值是 Date 类型，因此不能采用 DefaultNumberTextFieldCellEditor 或 DefaultTextFieldCellEditor，虽然 DefaultComboBoxCellEditor 理论上可以支持任何类型，但是我们不能枚举所有的时间。

这里我们着手来自己实现一个给开始时间属性编辑使用的日期编辑器。目前我们一直简

单的使用文本输入框来输入日期时间，因此我们可以继承DefaultTextFieldCellEditor来实现这个编辑器，就像继承DefaultNumberTextFieldCellEditor来实现数字编辑一样。因此参考DefaultNumberTextFieldCellEditor的实现方式，最终编写出如下代码：

DateCellEditor.as

```
package book.scheme{

import org.aswing.DefaultTextFieldCellEditor;
import org.aswing.JOptionPane;

/**
 * 日期时间单元格编辑器
 */
public class DateCellEditor extends DefaultTextFieldCellEditor{

    private var date:Date;

    public function DateCellEditor(){
        super();
    }

    /**
     * Subclass override this method to implement specified input restrict
     */
    override protected function getRestrict():String{
        return "0123456789 \\-";
    }

   /**
    * Sets the value of this cell.
    * @param value the new value of this cell
    */
    override protected function setCellEditorValue(value:*):void{
        var ccv:ColorCellValue = value;
        date = Utils.stringToDate(ccv.text);
        getTextField().setText(ccv.text);
    }

    /**
     * Subclass override this method to implement specified value transform
     */
    override protected function transforValueFromText(text:String):*{
```

```
        try{
            return Utils.stringToDate(text);
        }catch(e:Error){
            JOptionPane.showMessageDialog("提示", ""+e.message);
            return date;
        }
    }
}
}
```

可以看到，由于父类已经做了基本的文本输入编辑器的所有工作，读者可以阅读父类的源代码查看它做了那些工作，这里我们只需要覆盖 3 个方法，第一是返回输入字符限制，第二是设置编辑器要显示的值，第三是把编辑器上的字符串值转换为表格模型属性需要的值类型。由于我们前面给日期表格单元设置了转换器，为了支持颜色变化，我们用了 ColorCellValue 类型，因此在 setCellEditorValue 方法里，我们只能从它得到字符串类型的值，把他转换为 Date 类型，存储到成员变量 date 之中，到 transforValueFromText 方法的时候，它所做的工作实际上是把用户输入的日期字符串转换为 Date 类型。如果格式不正确的话，我们需要弹出对话框提示，并返回原来的值（setCellEditorValue 设置的值）以达到输入格式错误则不修改的目的。

单元格编辑的实现重点在于显示值（ColorCellValue，String）和实际值（Date）之间的转换关系，ColorCellValue 是表格单元格所用的值类型（由 *TaskManager.__dateTranslator* 转换而来），String 是单元格编辑器——继承自文本编辑器所用的值类型，Date 是表格数据模型——需要修改模型的属性所用的值类型。可以看出，DateCellEditor 实现，还是非常简单的，这是因为它的父类已经做了大部分工作。如果我们需要实现一个全新的编辑器，没有现成的能够提供基础工作的父类，那么实现将不会这么简单，不过只要掌握了三种值之间的转换关系，就能得心应手。并且，几乎所有单元格编辑器都可以继承自 AbstractCellEditor 类，它做了一些与具体编辑器显示不相关的所有工作，使得编辑器的编写可以稍微容易一些。

2.6 修饰和美化界面

目前我们的日程管理程序基本功能已经完全实现了，但是就界面来说，还显得不够大气，美观。这一节我们将重点介绍如何修饰和美化界面，内容包括：工具提示，边框，图标，前景/背景装饰器，元件装饰，自定义光标。

2.6.1 使用工具提示（JToolTip）

早在 2.3.1 节，我们就已经使用到了工具提示。在 CreateTaskPane.as 中的 labelHold 方法，我们用到了 *Component.setToolTipText* 方法给组件设置工具提示。同样，我们也可以用此方法给其他任何组件设置工具提示，通常，工具栏的按钮会采用图标（Icon）的形式，而文字提示则使用工具提示 ToolTip 的方式来显示。我们将在后面的章节讲图标，这一节先给工具栏的三个按钮设置工具提示，为 ToolBar 类的构造函数添加如下代码：

```
        addButton.setToolTipText("新建一个新的日程");
        removeButton.setToolTipText("删除选中的日程");
        editButton.setToolTipText("编辑选中的日程");
```

编译并运行程序，鼠标移动到新建日程按钮上停留片刻，我们将看到“新建一个新的日程”字样的工具提示，如下图：

（图 22）

本节知识点：

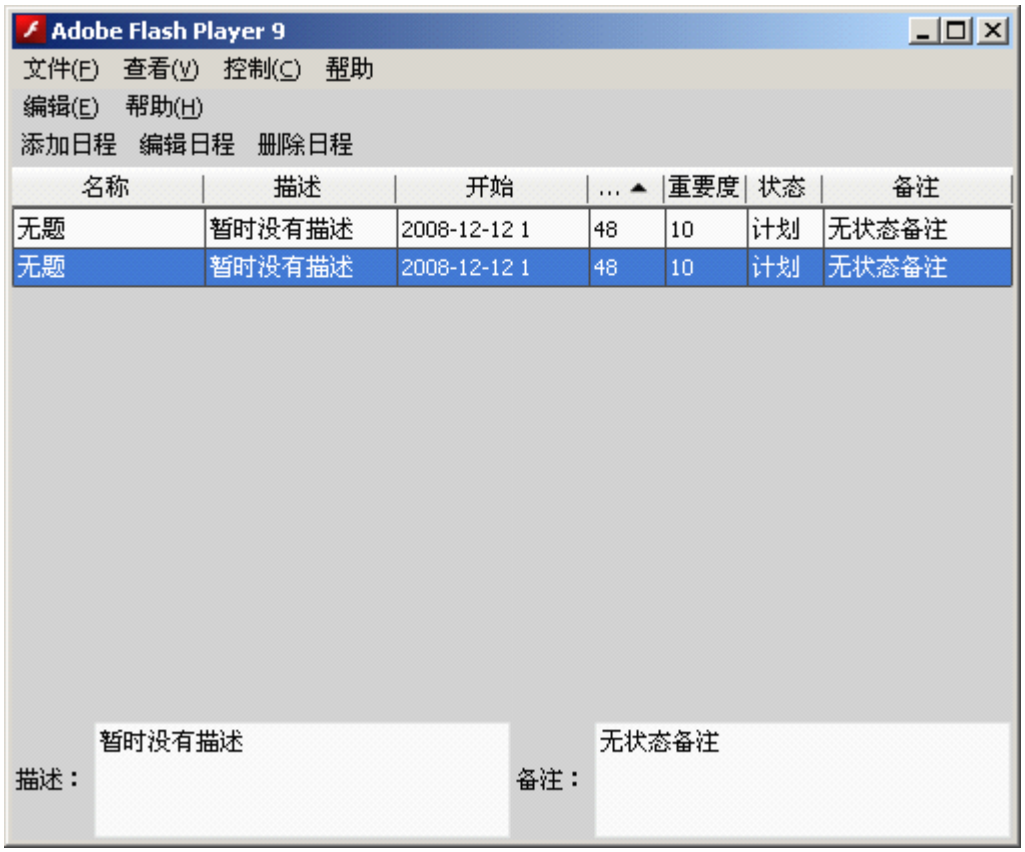
JToolTip：工具提示组件，通常并不直接创建此类的实例，而是采用 *Component.setToolTipText* 方法给组件设置提示文字，此方法的内部实现实际上是使用了一个共享的工具提示组件来显示提示，因为通常并不会有多个提示同时出现，所以采用一个共享的实例有助于节约内存。但是，当需要自定义工具提示的一些特性时，则需要自己创建甚至继承 JToolTip，JToolTip 是 Container 的直接子类，因此开发者还可以把它当作容器添加额外的内容。JToolTip 最重要的方法是 setTargetComponent，它用来设置工具提示的宿主，使得工具提示就明白在哪里显示，在何层显示，因此在创建 JToolTip 的时候，必须调用它设置宿主，然后工具提示才可能正确工作。此外，JToolTip 还可以设置提示框显示坐标是相对于组件位置还是当前鼠标位置，位置偏移量等，具体请查阅 api 文档。

JSharedToolTip：共享工具提示组件，由于工具提示通常并不需要同时显示多个，因此如果多个组件需要提示时，可以共用同一个工具提示，此类正是为此而诞生的。*Component.setToolTipText* 方法内部其实就是使用了此类来实现的。当需要给一个组件设置工具提示时，调用 registerComponent 方法注册，当需要移除时，调用 unregisterComponent。*JSharedToolTip.getSharedInstance* 静态方法始终返回同一个实例，因此需要全局共享工具提示组件时，即可调用此方法得到全局共享的实例。

2.6.2 背景色和边框(ASColor，Border)

我们程序目前的背景色全是采用的组件默认颜色，并且根组件采用的 JWindow 是没有背景色的，加上 JPanel 默认又是透明的，所以大部分底色是透到了影片的最底层的颜色——Flex 程序默认的蓝灰色。

这里我们尝试把几个主要 JPanel 面板都设置为不透明，让它拥有默认的背景色，给 Menu，TaskTable，TaskDetailsPane 都调用 *setOpaque(true)*，运行程序，我们将得到如下界面：



（图 23）

可以看到灰色背景替代了原来的蓝灰色，界面色调显得统一了。但是各大面板都是同一个颜色，显得层次不够分明。我们可以调用 Component.setBackground 方法来给不同的面板设置不同的背景色，把层次区分开来。经过随意的颜色的挑选，我们给 TaskTable，TaskDetailsPane 以及其内部的两个文本框设置了不同的几种颜色，效果如下：

（图 24）

可以看出较之原来层次分明了，但却并不美观，我可能需要一个外观设计师来帮助我选择配色方案，如果你在开发商用项目时面临此问题，一定需要找一个独具艺术感的设计师来协助，那样才能构造出美观的程序界面。但是编写此节时，并没有设计师协助我，希望读者可以忍受我这并不美观的配色。

虽然日程细节面板部分做了配色处理，但是层次依旧不够，它与中部的衔接太过紧密。如果我们需要让它独立起来，显得与其他部分有所区分，让用户一眼就可以看出它内部的几个组件组成了一个作用特定的独立整体，我应该怎样做呢？最常用的方法是使用边框框住这个部分，下面我们选用一个使面板看起来凹陷的边框，给 TaskDetailsPane 调用 *setBorder(new BevelBorder(null, BevelBorder.LOWERED));* 得到如下图效果：

（图 25）

可以看出，层次感又有所增强了。

本节知识点：

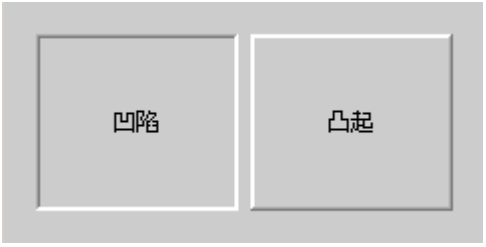
背景色：在 AsWing 中，每个组件都必有两个颜色属性，前景色和背景色，前景色通常代表组件的文字颜色，背景色代表底色，可以通过 setForeground 和 setBackground 来设置它们。与背景色相关联的还有不透明属性——opaque，有的组件默认是透明的，有的不是（视不同的 LookAndFeel 而定，后面章节会进行介绍），如果要确定背景色是否有效，可以调用 setOpaque 方法设定是否不透明。

ASColor：颜色类，含有 rgba 属性的类，此类为 immutable 特性的类，即一经创建，属性就不会再变化，与 String 类型一样。因此如果要改变一个组件的颜色，必须设置另一个颜色对象。不同的颜色对象，可能具有相同的颜色属性，比较两个颜色是否相同，可以用 ASColor.equals 方法来进行判断。

边框（Border）：组件的边框，通常大部分组件的默认边框都为 null，即没有边框。开发者可以通过给组件设置边框来修饰界面。Border 是一个接口，借此开发者可以实现任意自己想要的边框类型。通常，LookAndFeel 会给一些组件用设置边框的方式来进行修饰，这时如果你给这种组件设置了另外的边框，则会影响它原来的外观，你可以通过 *getBorder() is UIResource* 来判断现有边框是否为 LookAndFeel 的 UI 资源。实现自己的边框类方式和实现一个 Icon 类似，可参考下一节自定义 Icon 相关内容。

常用的边框：AsWing 自带有 7 个常用的边框类，分别是：

- **BevelBorder**：斜边边框。此边框通常用来构建凹陷或凸起效果，图示如下：

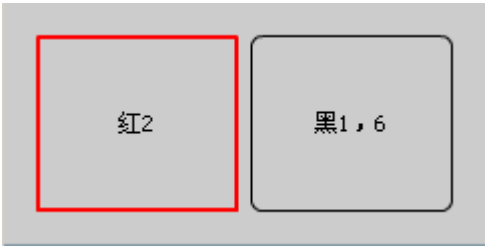


（图 26）

- **CaveBorder**：凹穴边框，此边框是由 TitledBorder 去掉标题文本简化而来的，它表现为一个高亮线条框，可以直角也可以圆角，图示如下：

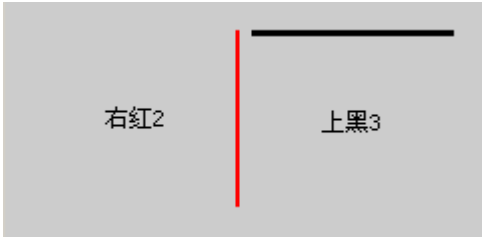
（图 27）

- **LineBorder**：线条边框，此边框表现为指定颜色，线条宽度，圆角跨距的线框，图示如下：



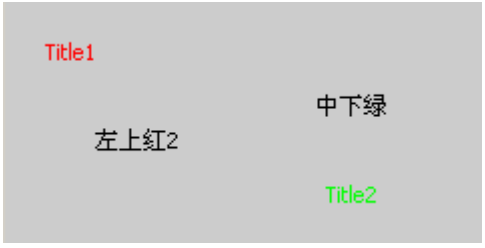
（图 28）

- **SideLineBorder**：单边线条边框，有时候只需要在组件的某一边有一条线，比如分割线，就可以使用此边框，图示如下：



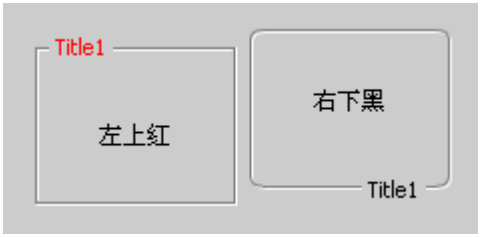
（图 29）

- **SimpleTitledBorder**：简单标题边框，此边框在组件上方或者下方显示一段文字，可作为一些组件面板的标题，图示如下：



（图 30）

- **TitledBorder**：带有标题和高亮线框的边框，这可能是最常用的，，图示如下：



（图 31）

- **EmptyBorder**：空边框，空边框看似什么都不做，其实用处非常广泛，因为它是空的，所以常常用于制造空隙，比如一个文本标签，它的期望大小是刚刚能完全显示文本字符串的大小，如果想让它上下左右拥有各 4 个像素的空白，那么可以给这个标签设置这样的边框 *new EmptyBorder(null, new Insets(10, 10, 10, 10))*。

这里没有详细介绍这几种边框的所有具体参数，由于边框类都比较简单，设置不同的属性值即可得到不同的效果，读者可以参阅 api 文档，自己试试不同的边框，不同的属性，各自不同的效果。这里给出前面示例的程序模板：

```
public class TestBorder extends Sprite{

    public function TestBorder(){

        AsWingManager.initAsStandard(this);

        var window:JWindow = new JWindow();
        var label1:JLabel = new JLabel("左上红");
        var label2:JLabel = new JLabel("右下黑");
        var pane:JPanel = new JPanel(
            new BoxLayout(BoxLayout.X_AXIS, 6));
        pane.setOpaque(true);
        pane.appendAll(label1, label2);
        pane.setBorder(new EmptyBorder(null,
            new Insets(16, 16, 16, 16)));
        window.setContentPane(pane);
        window.setSizeWH(240, 120);
        window.show();

        //由于JWindow没有背景色，这里设置一个填充色背景装饰
        window.setBackgroundDecorator(
            new SolidBackground(pane.getBackground()));

        var b1:TitledBorder = new TitledBorder(
            null,
            "Title1",
            TitledBorder.TOP,
            TitledBorder.LEFT, 8);
        b1.setColor(ASColor.RED);
        var b2:TitledBorder = new TitledBorder(
```

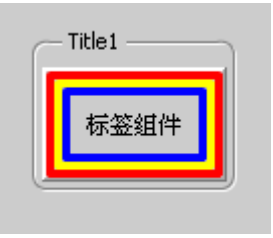
```
                null,
                "Title1",
                TitledBorder.BOTTOM,
                TitledBorder.RIGHT, 8, 6);
            b2.setColor(ASColor.BLACK);

            label1.setBorder(b1);
            label2.setBorder(b2);
        }
    }
```

读者可以尝试修改最后几行来测试不同的边框效果。

边框的嵌套：上面介绍的 AsWing 自带的边框类，都继承自 DecorateBorder 类，实现了装饰器模式（Decorator Design Pattern），边框可以嵌套使用。细心的读者可能已经发现，7 种边框构造函数的第一个参数 interior:Border 即代表内嵌边框，如果内嵌边框也是 DecorateBorder，那么它还可以再内嵌，用此方法可以组合成非常复杂的边框，比如，下一段代码：

```
var empty:EmptyBorder = new EmptyBorder(bevel, new Insets(6, 6, 6, 6));
var line3:LineBorder = new LineBorder(empty, ASColor.BLUE, 4);
var line2:LineBorder = new LineBorder(line3, ASColor.YELLOW, 4);
var line1:LineBorder = new LineBorder(line2, ASColor.RED, 4);
var bevel:BevelBorder = new BevelBorder(line1, BevelBorder.RAISED);
var title:TitledBorder = new TitledBorder(
    bevel,
    "Title1",
    TitledBorder.TOP,
    TitledBorder.LEFT, 8, 8);

label1.setBorder(title);
```

则可以形成如下边框：



（图 32）

最后再提醒一点，Border 只是一个接口，AsWing 自带的 7 种边框并不是所有，你可以根据自己的需要实现自己的边框，你甚至可以用一个资源图片来充当边框，只要你实现了 Border 接口的所有方法即可。

2.6.3 使用图标(Icon)

在本大节的第一小节中，我们提到过通常应该使用图标（Icon）使工具栏按钮显得更美观，这一节我们便来讲解图标的使用。

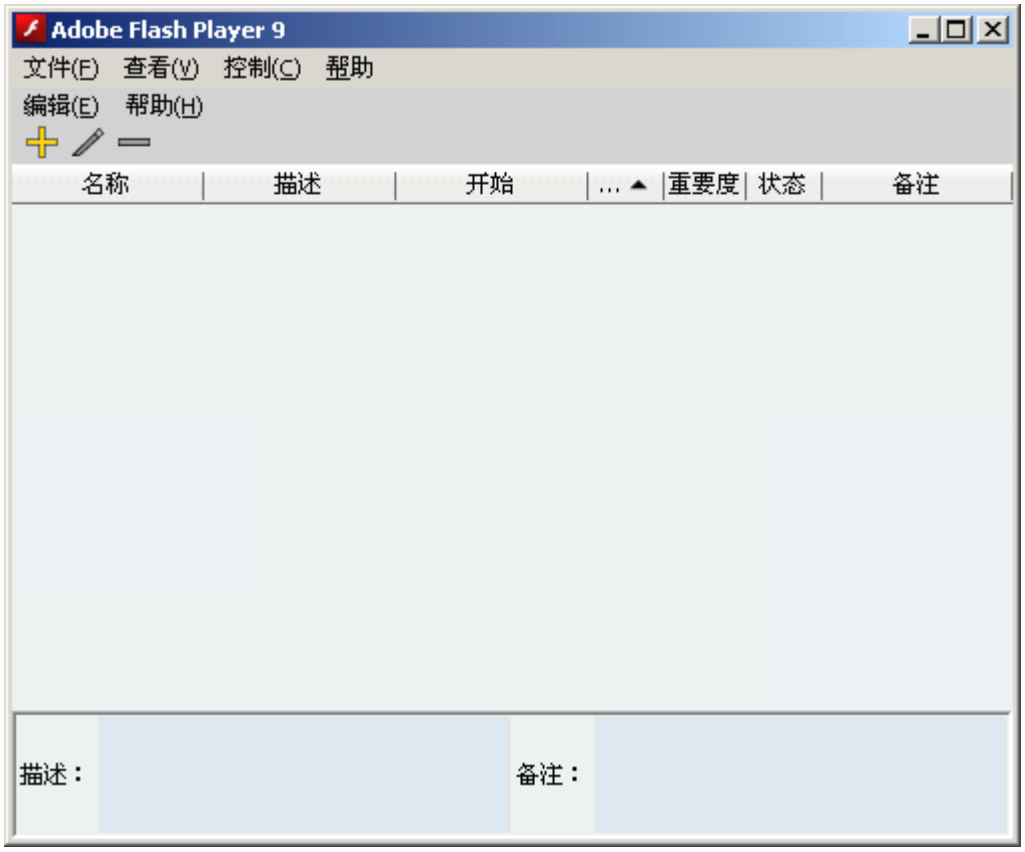
与 Border 相似的是 Icon 也是一个接口，因此开发者可以灵活的创造各种各样的实现。不同的地方是：Border 是所有组件都能使用的，Component 基类有 setBorder 方法，而 Icon 不是，Component 类并没有使用图标的方法。Icon 主要用于按钮（AbstractButton 的子类）以及其他特定需要图标的组件。

AsWing 中自带的 Icon 实现有 LoadIcon，AttachIcon 和 AssetIcon（前两者的父类）。LoadIcon 通过加载指定路径下的图片或者动画作为图标，AttachIcon 通过反射指定程序域（ApplicationDomain）下指定类名的元件作为图标，AssetIcon 通过指定元件实例作为图标。

我们给工具栏的三个按钮绘制图标并导出图片，以 LoadIcon 为例，给工具栏按钮设置图标，相关代码如下：

```
        addButton = new JButton(null,
            new LoadIcon("icons/add.png", 16, 16));
        removeButton = new JButton(null,
            new LoadIcon("icons/delete.png", 16, 16));
        editButton = new JButton(null,
            new LoadIcon("icons/edit.png", 16, 16));

        removeButton.setDisabledIcon(
            new LoadIcon("icons/delete2.png", 16, 16));
        editButton.setDisabledIcon(
            new LoadIcon("icons/edit2.png", 16, 16));
```

编译并运行程序，我们将得到如下界面：



（图 33）

JButton 继承自 AbstractButton，拥有一些列 setXXXIcon 方法，可以给按钮设置各个不同状态下的图标。构造函数和 setIcon 设置的 Icon 将作为默认 Icon 使用与所有状态，除非某个状态设置了自己专属的 Icon。比如上图中，我们看到编辑和删除按钮是灰色的图标，而实际上当有日程被选中时，这两个图标又会变成彩色（读者可以自己运行试试）。这是因为程序中我们给编辑和删除按钮调用了 setDisabledIcon 设置了不可用状态下的图标。

除了按钮，JFrame 可允许设置一个 Icon，作为左上角出现的窗体图标。另外 JLabel 也可以设置图标。

由于图标是非常重要且常用的组件装饰，这里我们讲讲如何自己实现一个 Icon。首先看看 Icon 的四个接口方法：

```
    function getIconWidth(c:Component):int;

    function getIconHeight(c:Component):int;

    function updateIcon(c:Component, g:Graphics2D, x:int,
y:int):void;

    function getDisplay(c:Component):DisplayObject;
```

前两者返回图标的宽高，方法传入的参数是图标的宿主组件（使用此图标的组件），这里返回的宽高是为了组件内部做布局所使用，如果宽高计算不正确，那么图标所呈现的图形可能会超越或不及应该达到的范围。第三个方法，更新图标，第一个参数为宿主组件，第二

个参数为宿主图形对象，后两个参数为图标应该出现的位置。最后一个方法则是返回图标元件。

通常，如果你的图标只需要在组件上使用绘图方法绘制一些图形，那么让 getDisplay 返回空，在updateIcon 里面调用 Graphics2D 的绘图方法绘制图形即可。如果你的图标需要用到特有的元件来支撑，那么 getDisplay 方法即返回这个元件实例。下面代码是一个绘制指定宽高的圆形红色填充图案的图标类代码：

```
public class CircleIcon implements Icon{

    private var width:int;
    private var height:int;

    public function CircleIcon(width:int, height:int){
        this.width = width;
        this.height = height;
    }

    public function getDisplay(c:Component):DisplayObject{
        return null;
    }

    public function getIconWidth(c:Component):int{
        return width;
    }

    public function getIconHeight(c:Component):int{
        return height;
    }

    public function updateIcon(c:Component, g:Graphics2D, x:int,
y:int):void{
        g.fillEllipse(new SolidBrush(ASColor.RED), x, y, width,
height);
    }

}
```

给一个 JLabel 调用 *setIcon(new CircleIcon(40, 40));*大概会得到如下画面效果：

（图 34）

可见实现一个图标并不复杂，只要你掌握了绘图函数，就能绘制出任何你想要的图标。但是，通常我们可能使用美术设计人员创建的图片或者 Flash 元件更多，因此也许你更需要在构造函数里创建一个元件，然后在 getDisplay 方法中返回它，这实现起来非常简单，

就不详述了，读者可以自行试验。

本节知识点：

图标（Icon）：在 AsWing 中，所有 AbstractButton 都能设置图标，包括按钮，单选复选框，菜单等等，并且不同的状态还可以设置不同的图标。图标接口共有 4 个方法，要实现自己的图标类并不难，不过通常使用 LoadIcon 或 AttachIcon 或者直接使用 AssetIcon 已能达到大部分的需要（与美术设计师创造的图形资源配合）。与 Border 不同的是，LookAndFeel 通常不会给组件设置图标来修饰组件（JFrame 的默认图标除外）。

AssetIcon：元件图标，通过一个元件实例创建的图标，图标元素就是传入的元件本身。构造函数形式为 AssetIcon(asset:DisplayObject=null, width:int=-1, height:int=-1, scale:Boolean=false)，第一个参数为图标的元件，它可以是任何 DisplayObject 对象，位图 Bitmap 或者动画 MovieClip 等都可以，width 和 height 指定图标的宽高，如果用默认值-1，那么它会采用元件的 width 和 height 属性，最后一个参数 scale，是指当指定的宽高和元件本身宽高不等时，是否需要缩放元件到指定的宽高值。

AttachIcon：绑定元件图标，它是 AssetIcon 的子类，通过指定元件类名和程序域创建元件实例，并用此实例作为图标元素。

LoadIcon：加载资源图标，也是 AssetIcon 的子类，通过加载指定路径下的一个元件资源来作为图标元素。

2.6.4 使用前景/背景装饰器(GroundDecorator)

由于 Border 专注于边框，而 Icon 又只是少部分组件的专利，那么 GroundDecorator 的出现将是对它们装饰范围的有力补充。每个组件都可以设置一个前景和背景装饰器（*Component.setBackgroundDecorator* 和 *Component.setForegroundDecorator* 方法），前景装饰器会被始终显示在最上层，背景装饰器会被始终显示在最底层。GroundDecorator 拥有强很大的装饰能力，通过它简单的接口定义即可看出：

```
    function updateDecorator(c:Component, g:Graphics2D,
b:IntRectangle):void;

    function getDisplay(c:Component):DisplayObject;
```

我们可以通过 updateDecorator 方法调用绘图接口绘制，也可以通过 getDisplay 方法返回一个显示元件来修饰。

需要注意的地方是，原则上，装饰器必须保证装饰图形处在 updateDecorator 方法传入的矩形范围之内（参数 b:IntRectangle），否则装饰图形可能会和组件的 Border 相交叠。当然如果你清楚组件的 Border，并且需要交叠效果，你也可以让装饰器覆盖整个组件范围。

AsWing 自带有 AssetBackground 和 SolidBackground 这两个装饰器实现，前者通过一个显示元件作为装饰资源，后者通过绘制一个纯色填充矩形来做修饰。读者可以查阅这两个类的源代码，它们的实现代码都非常简单，具体的处理代码不超过 5 行，可见实现一个装饰器是非常的简单。

前景/背景装饰器还有一个很有用的功能，即是让没有背景的组件，如 JRootPane，JPopup，JWindow，AssetPane，JLoadPane 等方便的拥有背景色。通过类似 *setBackground(new SolidBackground(yourColor))* 的方式既可以让它们由透明变为有色。在本大节第二小节，我们就已经用此方式给 JWindow 设置了背景色。

由于前景/背景装饰器的灵活，一些特制的 LookAndFeel（比如后面将会介绍的 SkinBuilderLAF）大量使用了它们作为组件外观装饰，因此如果你要给一个组件设置装饰器，需要注意是否会破坏它当前的外观。通常，LookAndFeel 不会给容器组件使用装饰器，特别是 JPanel 这样的纯容器。

由于装饰器的结构和工作原理与 Border 还有 Icon 相似，并且它的接口非常简单，很容易开发自己的实现。因此这里就不具体举例介绍了。读者可以自行编写测试程序进行试验。

本节知识点：

GroundDecorator：前景/背景装饰器，由于所有组件都可以拥有一个前景装饰器，一个背景装饰器，因此在装饰一个组件时，它是最好的选择，比如你要让一个窗口的底层显示一个动画，那么用 *yourWindow.setBackgroundDecorator(new AssetBackground(yourMovie))*;即可达到。

2.6.5 直接添加显示元件(DisplayObject)

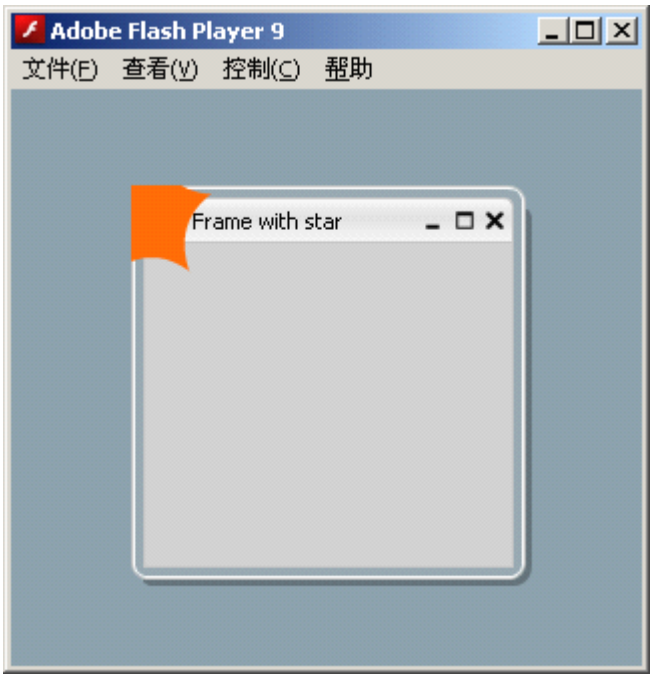
AsWing 组件可以随意的添加任何显示元件，这和 Flex 界面框架不同，Flex 界面框架要求被添加的显示元件必须实现 IUIComponent 接口，给开发人员与美术设计人员之间的配合带来了诸多不便。

要向 AsWing 组件中添加一个显示元件，可以简单的 addChild 即可，这是因为 AsWing 组件都是继承自 Sprite，并且对被添加的子元件无任何特殊要求，你完全可以把组件当成一个普通的 Sprite 来使用。比如，下面一段代码给一个 JFrame 左上角添加一个用 Flash CS3 绘制制作的红色星型 MovieClip，首先用 Flash CS3 创建文档 star.fla，在里面添加一个 MovieClip 元件，绘制星型图案，绑定元件的 Linkage 为 Star，类型为 MovieClip，然后发布 star.swf 影片文件，在 FlexBuilder 中编写代码如下：

```
public class TestStar extends Sprite{
```

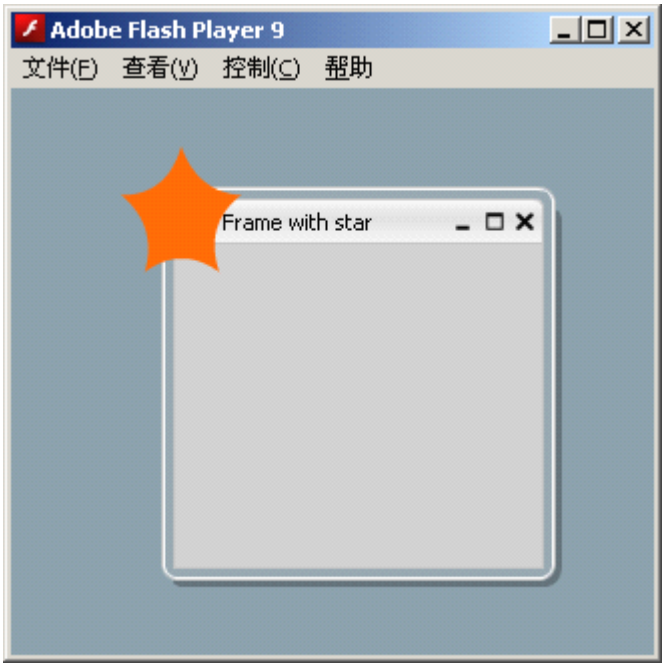
```
    [Embed(source="star.swf#Star")]
    private var StarClass:Class;

    public function TestStar(){

        AsWingManager.initAsStandard(this);

        var frame:JFrame = new JFrame(null, "Frame with star");
        //frame.setClipMasked(false);

        var star:DisplayObject = new StarClass() as DisplayObject;
        star.x = -20;
        star.y = -20;
        frame.addChild(star);

        frame.setSizeWH(200, 200);
        frame.show();
    }
}
```

编译并运行，将得到如下界面：



（图 35）

可以看到，星型图案出现了，并且不会影响组件的正常使用，不过星型图案被切掉了一部分，这是因为 AsWing 组件都默认有一个遮罩罩住组件范围，以防止组件的图形超出了自己的范围从而影响到其他组件。如果我们去掉程序中注释掉的那一句 *frame.setClipMasked(false);*即可使这个 JFrame 不采用遮罩，会得到如下界面：



（图 36）

可见，图形都显示出来了。

本节知识点：

直接添加显示元件：虽然 AsWing 组件没有限制往内添加任何显示元件，但是也有需要注意的地方。比如上例中关于遮罩的问题，和 JFrame 一样，所有组件默认都是被遮罩的，如果你的图形内容需要超出组件范围，那么就需要取消遮罩，但是要谨慎使用，取消遮罩也就意味着这个组件的图形可能会超出自己的范围从而与其他组件相叠。并且，对于 JFrame，取消掉遮罩还意味着最小化的时候组件的 contentPane 会暴露出来，因为遮罩失效了，最小化时 contentPane 不会被遮起。通常的解决方法是，要么对这样的 JFrame 取消最小化功能，要么监听最小化事件，在最小化之后把 contentPane 设置为不可见(setVisible(false))。另外，如果你添加的元件是一个 InteractiveObject，典型的比如一个绑定的 MovieClip，它会吃鼠标事件，为了不让其吃掉组件本应该吃的鼠标事件，你应该把它的 mouseEnable 和 mouseChildren 属性设置为 false，除非你故意需要它来吃事件。

2.6.6 使用自定义光标(Cursor)

AsWing 提供了自定义光标的管理类 CursorManager，使用此类可以设置自定义光标图形，隐藏/恢复系统光标等，甚至可以当鼠标处于某一个组件之上时使用一个特定的光标图形。

要显示一个自定义光标，可调用：

```
CursorManager.getManager(
    stage:Stage=null).showCustomCursor(
```

```
            cursor:DisplayObject,
            hideSystemCursor:Boolean=true);
```

getManager 静态方法首先获得一个 CursorManager 实例，CursorManager 会自动为每一个 Stage 创建一个实例，如果你的程序只存在一个 Stage（除了多窗口 AIR 程序，普通的 Flash 程序都只有一个 Stage）并且你在之前给 AsWingManager 初始化了 Stage（通过 AsWingManager.init 方法或者 AsWingManager.setRoot 方法），那么这里你可以简单的传入 null，意指使用初始化的 Stage 实例。

然后 showCustomCursor 第一个参数指定需要作为光标图案的元件实例，第二个参数指定是否同时需要隐藏系统光标。

要显示一个自定义光标，可调用：

```
    CursorManager.getManager(
        stage:Stage=null).hideCustomCursor(
            cursor:DisplayObject);
```

hideCustomCursor 方法会判断参数 cursor 是否是正在显示的自定义光标图案，如果是，那么移除它并恢复系统光标，如果不是则保持原状。

如果你要使鼠标处于某一个组件（甚至任何 InteractiveObject）之上时使用一个特定的光标图形，你可以调用更方便的方法：

```
    CursorManager.getManager(
        stage:Stage=null).setCursor(
            trigger:InteractiveObject,
            cursor:DisplayObject);
```

其中 trigger 参数为需要自定义光标的组件或任何 InteractiveObject，cursor 参数为需要显示的光标图案。当需要取消这个组件的自定义光标时，只需要再次调用此方法并给第二个参数传入 null 即可，如 setCursor(trigger, null)。

本节知识点：

自定义光标：光标管理器的方法都非常简单，本节就不再举例了，读者可以自己尝试编写例子来实验。这里讲讲自定义光标的内部实现细节，AsWing 自定义光标是通过在光标根容器中添加光标元件，并且监听鼠标移动事件来同步元件与鼠标的位置来实现的。光标根容器默认为 Stage，你也可以通过 setCursorContainerRoot 方法来设置到别的容器内，但注意鼠标根容器应该是处于一个比较根部的位置，以使得鼠标图形元件被添加时，能够始终显示在最上层。

2.6.7 包装 Flash IDE 创建的按钮（wrapSimpleButton）

这又是一个 AsWing 为了便于开发人员与美术设计人员协作而提供的特性。使用过 Flash IDE 的读者相信都知道，用 Flash IDE 创建一个特定形状的按钮，是非常的方便，

美术设计人员更是运用娴熟，因此，当你的程序需要一个美观的按钮时，你更希望直接使用美术人员创建好的按钮，而不是导出若干图片，一个一个的设置 Icon。

在这里，你只需要使用按钮的 wrapSimpleButton 方法，即可达到此目的。假如美术人员创建了一个按钮元件，在 Flash CS3 中绑定名为 MyButton，类型为 SimpleButton（使用 Flash CS3 创建的按钮都应该为此类型），那么在你的 AsWing 程序中，你可以这样使用：

```
//假设通过Embed这个按钮为MyButtonClass类
var myBtn:SimpleButton =
    new MyButtonClass() as SimpleButton;
var button:JButton = new JButton();
button.wrapSimpleButton(myBtn);
```

由此，button 将拥有你的美术设计师创建的按钮外观，并且各状态也完全保持一直。

不仅仅 JButton 拥有此能力， JToggleButton， JCheckBox 和 JRadioButton 也都拥有此能力，对于 JToggleButton，通过此方式，你可以非常方便的创建出类似 PhotoShop 里的眼睛  和锁  这样的开关按钮，SimpleButton 的 upState 将会作为 AsWing 按钮的普通状态， overState 将会作为 AsWing 按钮的 rollOver 状态，downState 则会作为 AsWing 按钮的按下状态（如果是 JToggleButton，JCheckBox 或 JRadioButton，也作为选中状态），不可用状态（disabled）采用灰度颜色来显示。

读者可以尝试创建一个 PhotoShop 的眼睛这样的选择按钮来进行实验。

本节知识点：

包装 SimpleButton：AbstractButton 及其子类都拥有此能力，此能力极好的方便了开发人员直接使用美术设计师创建的按钮元件。它的实现原理，其实是把 SimpleButton 按钮元件作为 AbstractButton 的 Icon，然后根据 AbstractButton 的状态变化，设置 SimpleButton 的状态，实现代码并不复杂，读者可以参阅 SimpleButtonIconToggle 类的源代码，观察它的内部实现细节。JButton 覆盖了 wrapSimpleButton 并采用了略微不同的实现方式，目的是使鼠标点击范围完全转移到 SimpleButton 上，使得包装后的 JButton 与原始 SimpleButton 的行为更一致。读者如果通过阅读相关代码，理解了包装 SimpleButton 的实现原理，那么你也就学会了更深入的修饰 AsWing 组件的方法了。

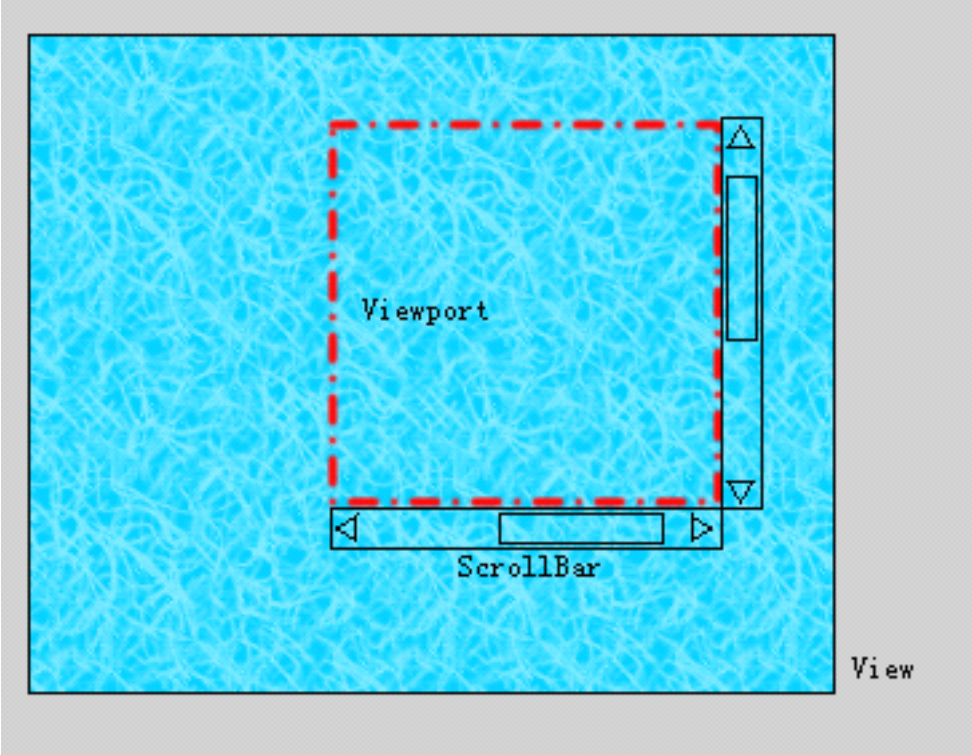
2.7 其他常用组件介绍

在上一节，我们的日程管理程序已基本告一段落，由于其功能不多，难以包含所有AsWing 常用组件，这一节我们将详细介绍前面未接触到的几个非常重要的组件，它们分别是滚动面板 JScrollPane，列表 JList，树 JTree，分页面板 JTabbedPane，滚动条 JScrollBar，滑动条 JSlider，等。

2.7.1 滚动面板（JScrollPane，Viewportable，JViewport）

早在 2.2.3 节，我们已经接触过 JScrollPane，并大概了解了其使用方法和工作原理。AsWing 组件对滚动面板的设计与不少流行的组件库设计理念不同，一些流行的组件库倾向于使大多数容器组件本身拥有滚动条能力，比如 Flex 组件的列表，表格，树等组件都是 ScrollControlBase 的子类，自动拥有滚动条的能力，也许这样会让开发者感觉使用更方便，但AsWing 不这样认为，AsWing 让开发者决定是否给某一个组件提供滚动条能力，绝对分离了滚动和容器的功能，减小了耦合，虽然在刚开始接触时，需要多了解一些使用方法，但是当你掌握了之后，会拥有更加灵活的能力。

JScrollPane 继承自 Container，它由两个滚动条（横竖各一个 JScrollBar）和一个视口（Viewportable）组成。滚动面板类本身并不控制内容的滚动，而是把这一工作分离给视口来完成，当滚动面板上的滚动条被用户拖动时，滚动面板根据滚动条的位置，调用视口设置视口中的内容位置，如下图所示：



（图 37）

图中虚线范围为视口（Viewport），右边和下边各为一个滚动条（ScrollBar），实线包含的内容为显示内容（View），实际运行的时候，View 只有视口透过的范围可见，视口范围之外的内容不可见，但滚动条被拖动时，View 随之移动，产生内容被滚动的效果。这就好比你给桌子挖一个方形的孔然后在桌子下面移动一本书。

JScrollPane 使用的视口可以是任何一个 Viewportable 的实现，比如 JViewport,JTextArea,JList,JTable,JTree,GridList 等。JScrollPane 的构造函数形式为：

```
JScrollPane(
        viewOrViewport:*=null,
        vsbPolicy:int=SCROLLBAR_AS_NEEDED,
        hsbPolicy:int=SCROLLBAR_AS_NEEDED)
```

第一个参数可以是一个 Viewportable 实例，将作为滚动面板的视口而工作；也可以是一个普通组件实例，此时 JScrollPane 将创建一个 JViewport 实例作为视口，并把此普通组件作为视口的显示内容（View）。

第二个参数指定垂直滚动条出现策略：

- **SCROLLBAR_AS_NEEDED** 指定当显示内容的高度大于视口高度时，才出现。
- **SCROLLBAR_NEVER** 指定始终不出现。
- **SCROLLBAR_ALWAYS** 指定始终出现。

第三个参数指定水平滚动条出现的策略，与垂直滚动条的各策略值道理相同，对应于显示内容和视口的宽度。

当然，除了构造函数，你也可以通过 JScrollPane 对应的共有方法来设置这些属性。并且，JScrollPane 还提供了设置/获取垂直和水平滚动条的方法，当你需要使用自定义

的滚动条，或者直接操控滚动条时，可以通过这些方法来进行。详见 JScrollPane 的源代码或 api 文档。

JViewport 是一个通用的视口组件，它可以接受任何组件作为显示内容（View）。它的构造函数形式为：

```
JViewport(
        view:Component=null,
        tracksWidth:Boolean=false,
        tracksHeight:Boolean=false)
```

第一个参数指定这个视口的显示内容。

第二个参数指定显示内容是否始终强制保持与视口的宽度相等——即不需要水平滚动。

第三个参数指定显示内容是否始终强制保持与视口的高度相等——即不需要垂直滚动。

通常，除了构造函数，你也可以通过相关共有方法来设置这些属性。通过 setViewPosition 等方法，你还可以直接控制显示内容的滚动位置。

Viewportable 接口定义了一系列方法来保证视口的工作能力，在 AsWing 自带组件中，实现了此接口的类分别为 JViewport，JTextArea，JList，JTable，JTree，GridList 等，其中除了 JViewport 可以自由接受设置显示内容，其它实现均不可设置显示内容，因为它们都将自身的内容作为显示内容，比如 JTextArea，它的显示内容就是自身的文本内容。其实，从 Viewportable 提供接口中并没有包含 setView 方法这一点可以看出，视口的实现并不要求视口与显示内容分离，通常大多数视口和显示内容都是一个整体，自己管理自己的内容显示。

例子，如下代码创建了一个 JScrollPane 使用一个 JViewport，显示内容为一个 JLoadPane 加载一个图片。除了 JScrollPane 自身的滚动条之外，还额外提供了两个按钮来滚动显示内容。代码如下：

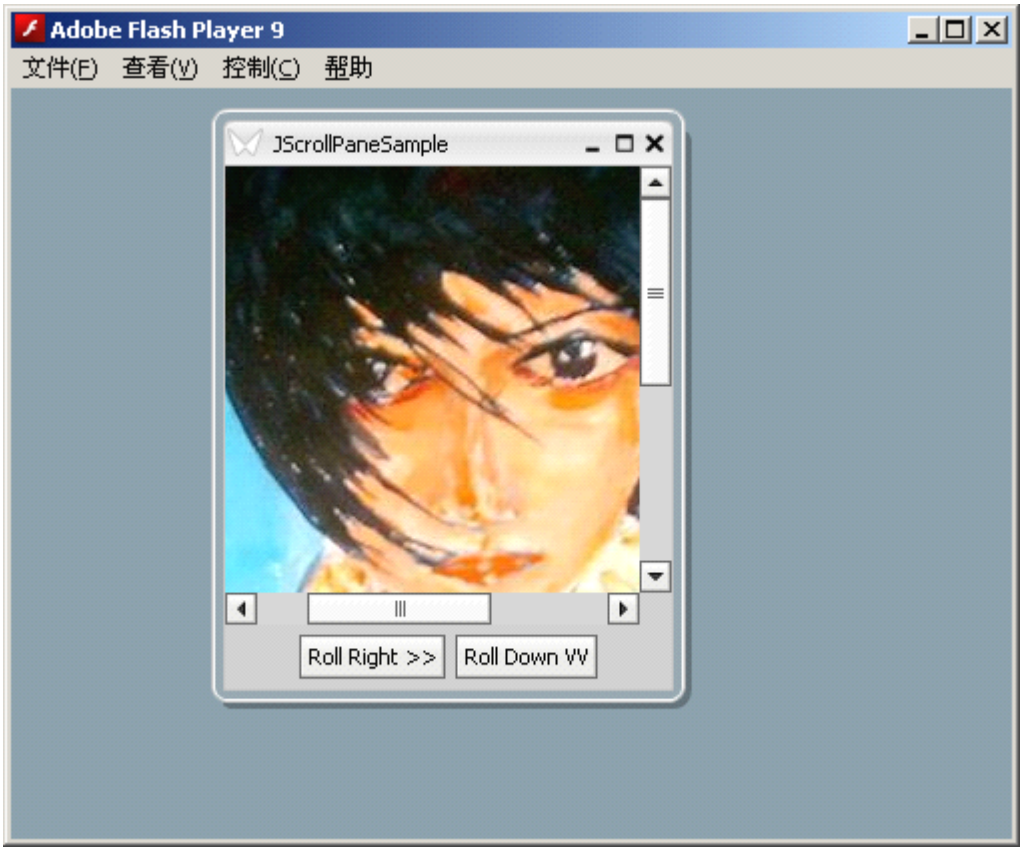
JScrollPaneSample.as

```
package{

import flash.display.Sprite;
import flash.events.Event;
import flash.net.URLRequest;

import org.aswing.AsWingManager;
import org.aswing.BorderLayout;
import org.aswing.FlowLayout;
import org.aswing.JButton;
import org.aswing.JFrame;
import org.aswing.JLoadPane;
import org.aswing.JPanel;
import org.aswing.JScrollPane;
import org.aswing.JToggleButton;
import org.aswing.JViewport;
```

```
public class JScrollPaneSample extends Sprite{

    private var scrollPane:JScrollPane;
    private var viewport:JViewport;
    private var loadPane:JLoadPane;
    private var rollRightButton:JButton;
    private var rollDownButton:JButton;

    public function JScrollPaneSample(){
        super();
        AsWingManager.initAsStandard(this);
        var dialog:JFrame = new JFrame(null, "JScrollPaneSample");
        loadPane = new JLoadPane();
        viewport = new JViewport(loadPane);
        scrollPane = new JScrollPane(viewport);

        rollRightButton = new JButton("Roll Right >>");
        rollDownButton = new JButton("Roll Down VV");

        var pane:JPanel = new JPanel(new BorderLayout());
        pane.append(scrollPane, BorderLayout.CENTER);
        var buttons:JPanel = new JPanel(new
FlowLayout(FlowLayout.CENTER));
        buttons.appendAll(rollRightButton, rollDownButton);
        pane.append(buttons, BorderLayout.SOUTH);

        dialog.setContentPane(pane);
        dialog.setSizeWH(240, 300);
        dialog.show();
        loadPane.load(new URLRequest("paint1.jpg"));

        rollRightButton.addActionListener(__rollRight);
        rollDownButton.addActionListener(__rollDown);
    }

    private function __rollRight(e:Event):void{
        viewport.setViewPosition(
            viewport.getViewPosition().move(40, 0));
    }

    private function __rollDown(e:Event):void{
        viewport.setViewPosition(
            viewport.getViewPosition().move(0, 40));
```

```
    }
}
}
```

编译并运行后将得类似到如下的效果：



（图 38）

操作后可以发现，当点击下方的按钮使显示内容滚动时，上方的滚动条也会同步滚动。这表明 JScrollPane 会自动探测到视口显示内容的位置变化并同时更新滚动条的滚动值，这自动完成的效果，是非常合理并对用户友好的。

2.7.2 列表（JList，VectorListModel）

列表组件是非常常用的组件，但是由于功能局限，我们的日程管理程序并没有使用到这一组件，因此在这里单独讲解。

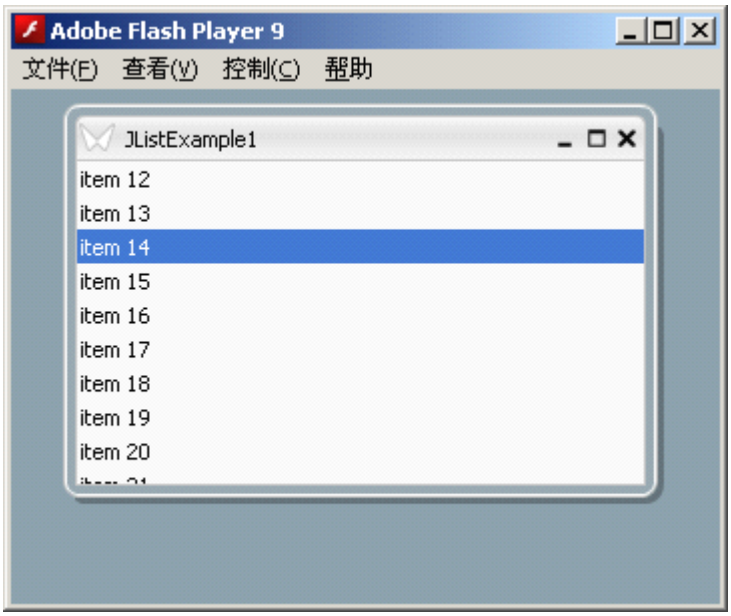
我们首先看下面这个简单的列表例子程序：

JListExample1.as

```
package{

import flash.display.Sprite;
```

```
import org.aswing.AsWingManager;
import org.aswing.BorderLayout;
import org.aswing.JFrame;
import org.aswing.JList;
import org.aswing.VectorListModel;

public class JListExample1 extends Sprite{

    private var model:VectorListModel;
    private var list:JList;

    public function JListExample1(){
        super();
        AsWingManager.initAsStandard(this);

        model = new VectorListModel();
        for(var i:int=0; i<100; i++){
            model.append("item " + i);
        }
        list = new JList(model);

        var frame:JFrame = new JFrame(this, "JListExample1");
        frame.getContentPane().append(list);
        frame.setSizeWH(300, 200);
        frame.show();
    }

}
}
```

编译运行它将得到如下界面：

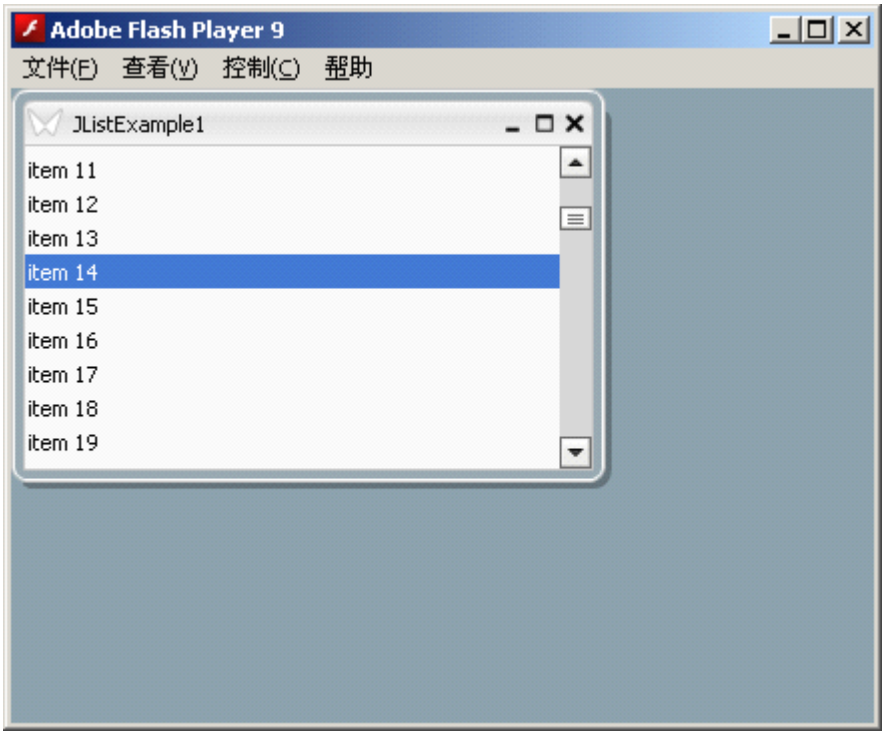


（图 39）

可以看到，列表如期显示出来了，并且能随着鼠标滑轮的滚动而滚动，不过奇怪的是，总条目大于显示出来的条目，却看不到滚动条的出现。哈哈，对了，在前面的章节我们已经讲过很多次了，除了 JScrollPane，AsWing 自带的任何其他组件都不会自动出现滚动条（除非你利用 JScrollBar 制作一个这样的新组件）。由于 JList 实现了 Viewportable，因此我们简单的修改上面的第 26 行为：

```
        frame.getContentPane().append(new JScrollPane(list));
```

则会得到如下含有滚动条的界面：



（图 40）

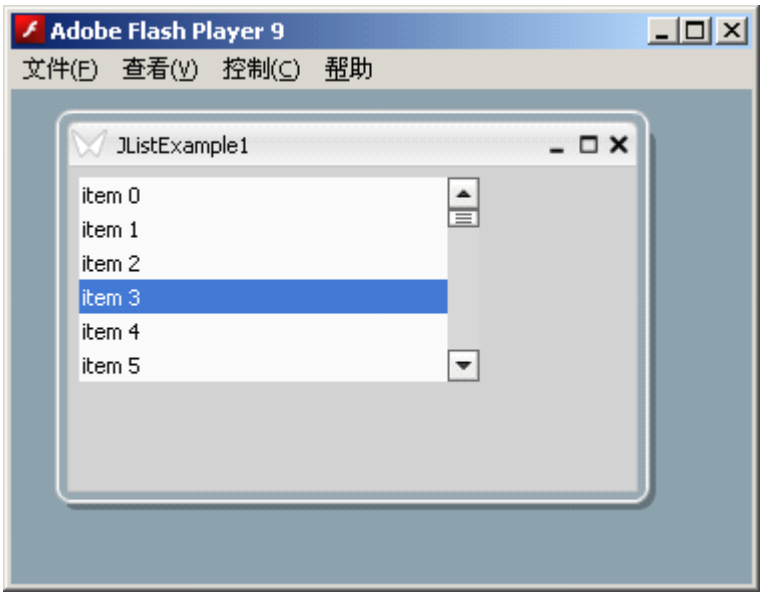
JScrollPane 的具体原理可回顾上一节，这里就不多述了。

JList 与前面我们详细讲解过的 JTable 类似，数据由一个 Model 来提供，在上面的例子中，我们创建了一个 VectorListModel 实例来做列表的 Model。列表的 Model 必须实现 ListModel 接口，ListModel 接口相对 TableModel 来说要简单很多，因此 AsWing 内部也只提供了 VectorListModel 这一个实现方案，大多数情况下，我们用它就足够了。

上面我们是采用 JFrame 的 contentPane 的默认的布局（BorderLayout），并把包含了 JList 的 JScrollPane 默认居中显示，因此它会忽略自己的期望大小而填充整个 contentPane，如果我们把 contentPane 设置布局管理器为 FlowLayout，则会得到如下的界面：

（图 41）

这是因为 JList 在默认情况下难以计算自己的期望大小，因此产生了一个不够合理的值，使得这界面效果根本就不可用。对于这种情况，我们要么给包含它的 JScrollPane 调用 setPreferredSize 手动设定一个合适的期望大小，要么调用 JList 的 setVisibleCellWidth 和 setVisibleRowCount 方法设定期望可视的宽度和高度。比如，我们设这两个值分别为 200 和 6，那么则会得到如下界面：



（图 42）

这两个值，前者告诉 JList 期望显示宽度为 200，后者告诉 JList 期望显示出来的条目为 6 个。JList 根据这些信息，就可以计算出自己期望的尺寸大小了。

同样，和 JTable 类似，JList 也可以自定义单元格 Cell，通常只需要实现自己的 ListCell 即可达到自定义的效果。这里我们以一个图片列表为例：假设一个列表，需要在单元格里面显示图片和图片名，由此，我们可以创建如下单元格类：

LoadImageListCell.as

```
package{

import flash.events.Event;
import flash.net.URLRequest;

import org.aswing.ASColor;
import org.aswing.AssetPane;
import org.aswing.BorderLayout;
import org.aswing.Component;
import org.aswing.JLabel;
import org.aswing.JList;
import org.aswing.JLoadPane;
import org.aswing.JPanel;
import org.aswing.ListCell;
import org.aswing.border.LineBorder;
import org.aswing.geom.IntDimension;

public class LoadImageListCell implements ListCell{

    private var pane:JPanel;
    private var loadPane:JLoadPane;
    private var label:JLabel;
    private var value:JListExample2Value;

    public function LoadImageListCell(){
        pane = new JPanel(new BorderLayout());
        loadPane = new JLoadPane();
        loadPane.setScaleMode(AssetPane.SCALE_FIT_PANE);
        //这里必须预先给loadPane设置期望大小
        loadPane.setPreferredSize(new IntDimension(64, 64));
        loadPane.addEventListener(Event.COMPLETE, __loadComplete);
        label = new JLabel();
        pane.append(loadPane, BorderLayout.CENTER);
        pane.append(label, BorderLayout.EAST);
        pane.setBorder(new LineBorder());
        //设置为透明策略，为使不同状态不同背景色有效
```

```
        pane.setOpaque(true);
        label.setOpaque(false);
        //loadPane不需要设置透明策略，因此它是没有底色的组件
    }

    private function __loadComplete(e:Event):void{
        //图片加载完成后，调用此方法使图片得到布局
        loadPane.doLayout();
    }

    public function setCellValue(v:*):void{
        value = JListExample2Value(v);
        loadPane.load(new URLRequest(value.getImage()));
        label.setText(value.getLabel());
    }

    public function getCellValue():*{
        return value;
    }

    public function setListCellStatus(list:JList, selected:Boolean,
index:int):void{
        if(selected){
            pane.setBackground(list.getSelectionBackground());
            pane.setForeground(list.getSelectionForeground());
        }else{
            pane.setBackground(list.getBackground());
            pane.setForeground(list.getForeground());
        }
        label.setFont(list.getFont());
    }

    public function getCellComponent():Component{
        return pane;
    }

}
}
```

此单元格组件为一个 JPanel，其中包含一个 JLoadPane 来加载图片，一个 JLabel 来显示图片名。

这里有不少需要注意的地方，首先 JLoadPane 在创建之后就设死了期望大小，这是因为 ListCell 需要在设置 value 之后立刻返回期望大小，从而让 JList 能够马上布局显示出来的条目，而 loadPane 默认情况下返回的期望大小是与加载的图片大小相关，且加

载图片需要一定时间，所以这里只能预先设死它，不需要它在加载图片后再计算（因为那时再计算，已经没有用了）。其次，我们监听了 JLoadPane 的加载成功的事件，并在事件处理函数中调用 doLayout 方法使得图片得到布局，通常情况下，我们是不需要这么做的，只有在 JLoadPane 处于 Cell 中，才不得已这样做，这是因为 JList 接管了它内部单元格的 validate 周期，在给每个 Cell 设置 value 之后，JList 会立即 validate 每个 Cell 的组件，Cell 构成组件自身的 revalidte 调用不再生效，所以需要在 JLoadPane 加载成功后手动调用 doLayout（通过 validate 来间接调用 doLayout 也可以）。还有，这里把 JPanel 设置为不透明，JPanel 内部的组件都设置为透明，然后在 setListCellStatus 中，改变 JPanel 的背景色，使得单元格在状态改变时，可以显示出不同的背景色。

在这个 Cell 实现里，我们只接受 JListExample2Value 类型的数据，此类定义如下：

JListExample2Value.as

```
package{

public class JListExample2Value{

    private var image:String;
    private var label:String;

    public function JListExample2Value(image:String, label:String){
        this.image = image;
        this.label = label;
    }

    public function getImage():String{
        return image;
    }

    public function getLabel():String{
        return label;
    }

    public function toString():String{
        return "JListExample2Value(image:" + image + ", label:" +
label + ")";
    }
}
}
```

相应的，在主类的 VectorListModel 中，就必须填入此类型的数据，主类编写如下：

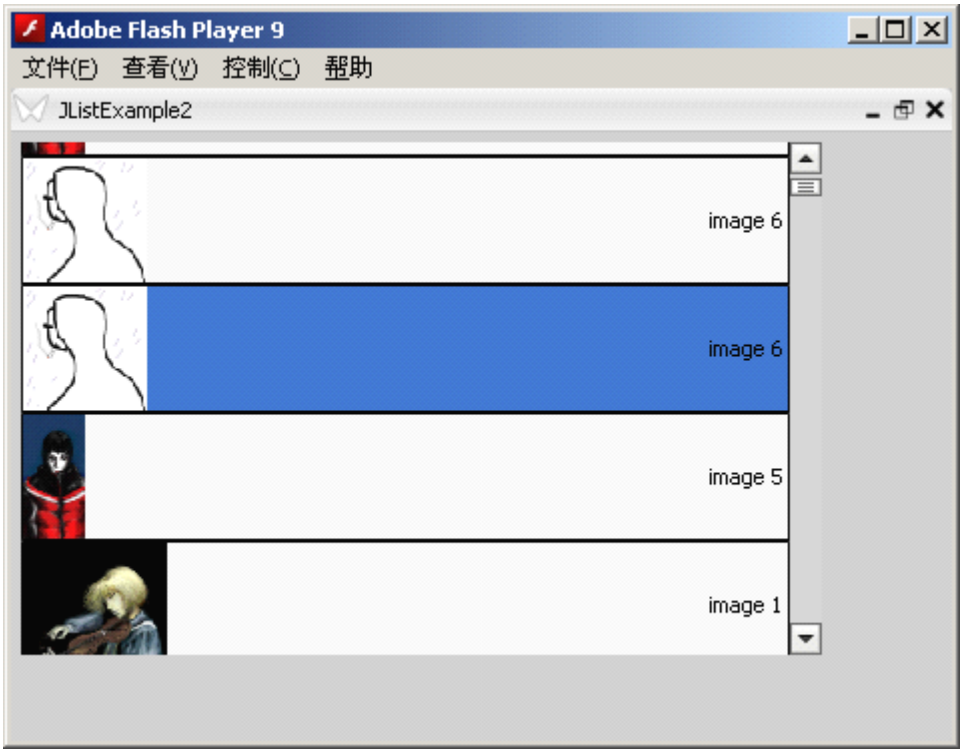
JListExample2.as

```
package{

import flash.display.Sprite;
```

```
import org.aswing.AsWingManager;
import org.aswing.FlowLayout;
import org.aswing.GeneralListCellFactory;
import org.aswing.JFrame;
import org.aswing.JList;
import org.aswing.JScrollPane;
import org.aswing.VectorListModel;

public class JListExample2 extends Sprite{

    private var model:VectorListModel;
    private var list:JList;

    public function JListExample2(){
        super();
        AsWingManager.initAsStandard(this);

        model = new VectorListModel();
        for(var i:int=0; i<100; i++){
            var index:int = int(Math.random()*7+1);
            model.append(new JListExample2Value(
                "image"+index+".png",
                "image " + index));
        }
        list = new JList(model,
            new GeneralListCellFactory(
                LoadImageListCell,
                true,
                true,
                64
            ));

        list.setVisibleCellWidth(400);
        list.setVisibleRowCount(4);

        var frame:JFrame = new JFrame(this, "JListExample2");
        frame.getContentPane().setLayout(new FlowLayout());
        frame.getContentPane().append(new JScrollPane(list));
        frame.setSizeWH(400, 300);
        frame.setState(JFrame.MAXIMIZED);
        frame.show();
    }
```

```
}
}
```

编译并运行此程序，将得到如下界面：



（图 43）

操作此界面时会发现一个奇怪的现象，在滚动和选择条目时，图片都会闪烁。这是因为我们为 GeneralListCellFactory 指定了单元格需要共享实例，要让有限的单元格实例显示众多的条目，JList 需要每次界面变化都给单元格设置最新对应的数据（调用 setCellValue），因此在界面操作时单元格会频繁的加载图片，从而造成如此的不友好状况。如果我们给 GeneralListCellFactory 构造函数第二个参数设为 false，重新编辑并运行程序，会发现程序的启动会慢一点点，但是之后的界面操作，不会有闪烁现象，而且滚动时也非常流畅。这是因为启动的时候，由于不共享单元格，因此创建了和 Model 里面条目数量一样多的单元格实例（此例为 100 个），并且会几乎同时加载 100 张图片，因此启动时间稍长，但是之后，每次操作，都不会影响到单元格的数据，不会重新加载图片，从而变得非常流畅。

读者还可以尝试修改 GeneralListCellFactory 构造函数其他几个参数，以观修改后的效果。

这个例子的单元格，我们用了 JPanel+JLoadPane+JLabel 的方式，略显复杂，目的是为了揭露 JList 和单元格之间的比较隐晦的部分特性，比如单元格组件的 validate 周期被劫持，单元格期望大小需要在 setCellValue 之后立刻能返回正确的值，等等。如果仅仅是为了实现图片+文字的单元格功能，使用 JLabel+LoadIcon 的方案也许会更简单，读者可以尝试修改 LoadImageListCell，把加载图片的任务交给 JLabel 的 LoadIcon，文字显示由 JLable 自身承载，实现与本例程序相同的功能。

JList 是一个实现了 Viewportable 的组件，用来显示一个从上到下排列的数据列

表，条目单元的显示由 ListCell 定义，ListCell 的实例由 ListCellFactory 产生，数据模型由 ListModel 提供。JList 拥有的事件比 JTable 略多，除了条目选择变化事件外，还拥有一些列与条目自身相关的事件，比如条目被点击，条目在鼠标移出范围时等，注意，如果要监听条目双击事件，需要给单元格组件（getCellComponent 方法返回的组件）设置 doubleClickEnabled=true，否则不会生效。详细的事件定义请参阅相关 api 文档。

VectorListModel 是 AsWing 自带的 ListModel 实现，它的行为类似一个 Vector，可以随意在任何位置插入/删除数据，并且在数据改变时，会自动触发对应的事件，使得 JList 自动更新界面显示。读者可以尝试给上面的例子程序加入删除/添加条目的功能，观察 JList 是否会自动的更新。

ListCell 定义了 JList 的单元格渲染，和 TableCell 原理类似，AsWing 自带的实现有 DefaultListCell，它表现为把数据以字符串的方式显示（数据的 toString() 方法的返回值将作为显示字符串）。

ListCellFactory 是创建 ListCell 的工厂，与 TableCellFactory 类似，但它还负责更多的事情，除了创建 ListCell 的方法外，它还有：

- ***isAllCellHasSameHeight():Boolean*** 方法，指出此工厂生产的单元格是否都以相同的高度来显示。如果此方法返回 true，那么 getCellHeight 返回的值将作为所有单元格的高度值。
- ***getCellHeight():int*** 方法，返回所有单元格应该被设置的高度值。仅当 isAllCellHasSameHeight 方法返回 true 时有效。
- ***isShareCells():Boolean*** 方法，指出是否应该共享使用单元格实例，如果此方法返回 true，JList 会只创建能够被显示出来的单元格数量（比如 JList 高度为 100，单元格高度为 20，那么单元格实例只会被创建出 5 个左右），在 JList 滚动的时候，实时给单元格设置滚动后对应的数据；如果此方法返回 false，JList 会为每个数据条目创建一个单元格，这样在 JList 滚动的时候，就不用再次设置各单元格的数据，只需要调整单元格组件的位置即可。注意，在共享单元格的时候，isAllCellHasSameHeight 返回 false 无效，会始终视为所有单元格都拥有 getCellHeight 返回相同的高度值。

PreferredSize，关于期望大小，尽管本节我们的例子都给 JList 设置了 VisibleCellWidth 和 VisibleRowCount，但是在通常的程序开发中，笔者很少这样做，因为通常我们都需要让 JList 这样的大组件占满剩余的区域（处于 BorderLayout 的 Center 位置），由此以来，期望大小就不用在乎，而且，对于非共享 Cell 的情况下，JList 会为每个 Cell 计算各自的期望宽度，并根据最大的宽度来作为期望宽度，而且，在需要水平滚动条的时候，则能够自适应出现合理的水平滚动条。

之所以让 ListCellFactory 有这样的参数，是为了效率和灵活性，JList 拥有比 JTable 更大的变化性。比如，上例我们就不能共享单元格，否则每次操作都会重新加载图片，无论是对网络压力还是用户感受，都不利。再者，如果一个列表，含有上万级别的条目数，就最好是采用共享单元格实例的方案，有助于节约内存。另外，允许单元格拥有不同高度的策略，使得表格能够有更大的变化性，比如可以方便地实现类似 QQ2008 风格的好友

列表（可参考文章 http://www.aswing.org/?p=232）。

提示：*关于单元格组件的 validate 周期被劫持这一特性，在 JTable 和 JTree 这类需要 Cell 来渲染条目的组件中也存在，因此在实现任何大型组件的单元格 Cell 的时候，都需要注意这一点。*

2.7.3 树（JTree，TreeModel）

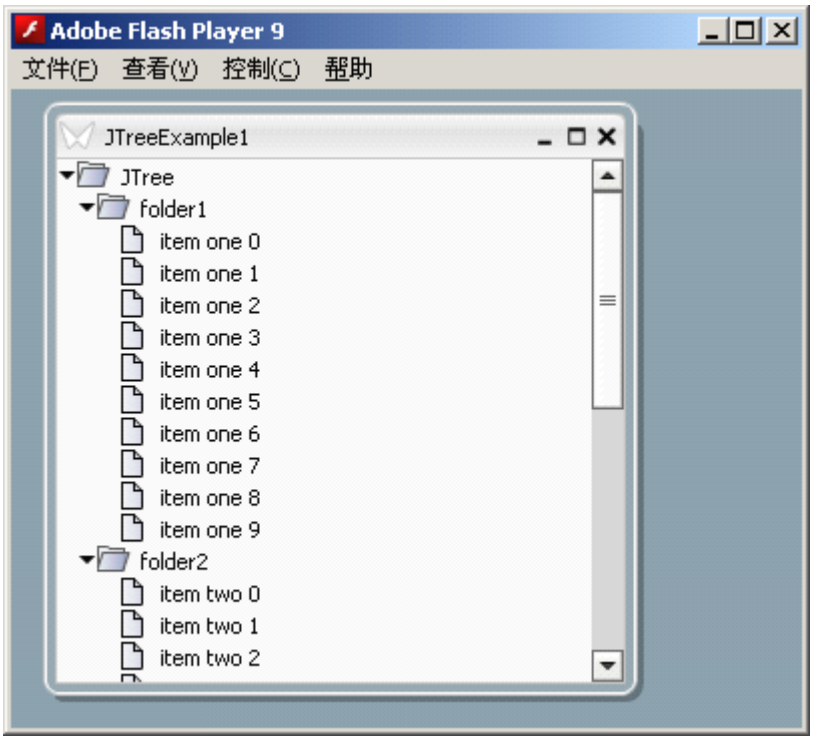
树 JTree 组件，用于表现树状结构的数据，也是非常常用的组件，树组件和 JList/JTable 一样属于比较复杂的组件，我们这里也从一个最简单的例子开始，见如下代码：

JTreeExample1.as

```
package{

import flash.display.Sprite;

import org.aswing.AsWingManager;
import org.aswing.BorderLayout;
import org.aswing.JFrame;
import org.aswing.JPanel;
import org.aswing.JScrollPane;
import org.aswing.JTree;
import org.aswing.tree.DefaultMutableTreeNode;
import org.aswing.tree.DefaultTreeModel;

public class JTreeExample1 extends Sprite{

    private var tree:JTree;
    private var frame:JFrame;

    public function JTreeExample1(){
        super();
        AsWingManager.initAsStandard(this);

        frame = new JFrame(this, "JTreeExample1");
        var pane:JPanel = new JPanel(new BorderLayout());

       var root:DefaultMutableTreeNode =
        new DefaultMutableTreeNode(createItem("JTree"));
        var parent:DefaultMutableTreeNode;
```

```
        parent = new DefaultMutableTreeNode(createItem("folder1"));
        root.append(parent);
        var i:int;
        for(i=0; i<10; i++){
            parent.append(new   DefaultMutableTreeNode(createItem("item
one " + i)));
        }

        parent = new DefaultMutableTreeNode(createItem("folder2"));
        root.append(parent);
        for(i=0; i<20; i++){
            parent.append(new   DefaultMutableTreeNode(createItem("item
two " + i)));
        }

        parent = new DefaultMutableTreeNode(createItem("folder3"));
        root.append(parent);
        for(i=0; i<30; i++){
            parent.append(new   DefaultMutableTreeNode(createItem("item
three " + i)));
        }

        var model:DefaultTreeModel = new DefaultTreeModel(root);

        tree = new JTree();
        tree.setModel(model);
        pane.append(new JScrollPane(tree), BorderLayout.CENTER);
        frame.setContentPane(pane);

        frame.setSizeWH(300, 300);
        frame.show();
    }

    private function createItem(value:String):*{
        return value;
    }
}
}
```

编译并运行此程序，将得到类似如下的界面：

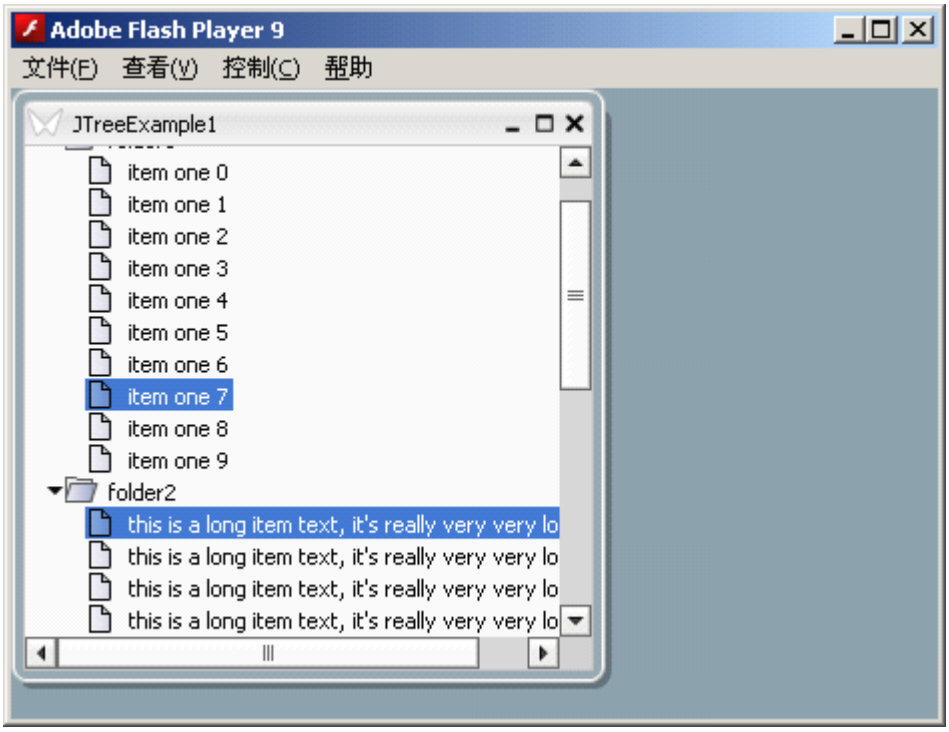


（图 44）

与 JList 类似，JTree 的期望大小也需要通过一些列方法的设置来进行调整，它们是：

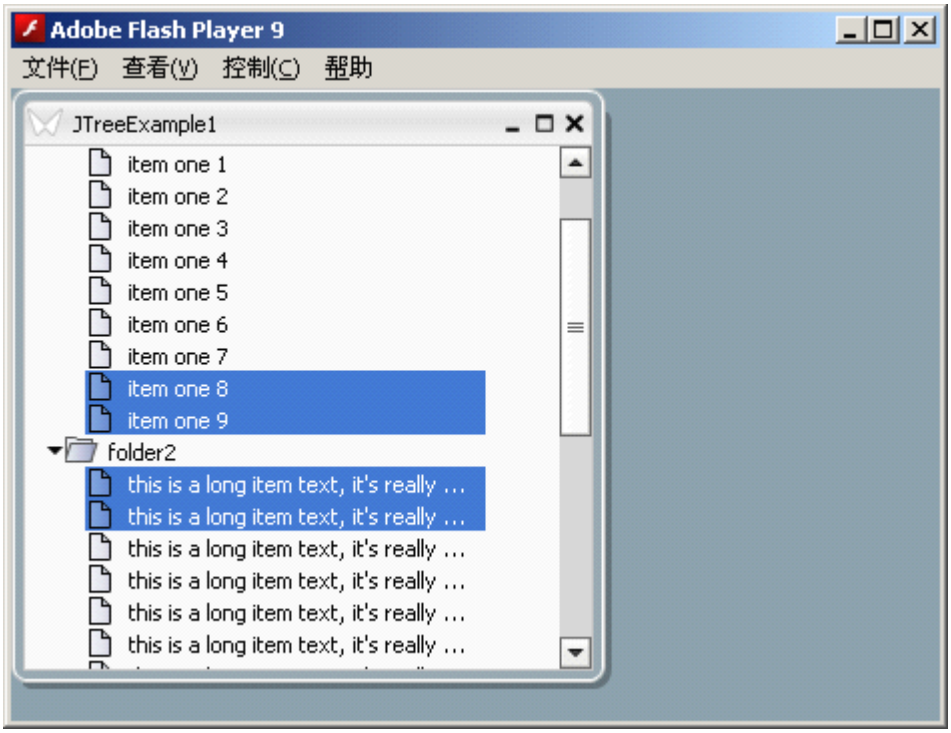
- ***setRowHeight(rowHeight:int):voia***，设置每一行的高度（默认值为 16），JTree 的所有行都拥有相同的高度。这与 JList 不同，JList 为了拥有更大的灵活性，把此设定交给了 ListCellFactory，并且允许不同的行可以有不同的高度，JTree 由于其自身的复杂性，没有提供这样的灵活性，采用这一更利于管理和运行效率的统一高度的策略。
- ***setVisibleRowCount(newCount:int):voia***，设置期望显示出的行数，它的功能与 JList 的同名函数相同。
- ***setFixedCellWidth(width:int):voia***，设置-1 指明自适应条目的期望宽度，设置非 -1 值指明所有条目采用指定宽度值。此方法与 JList 的 setVisibleCellWidth 方法含义不完全相同，JList 由于在共享 Cell 的时候，无法计算宽度，所以需要设置（非共享 Cell 时可以不用设置即可自适应），而JTree 能够计算所有条目的期望宽度，所以除非你想使所有条目拥有共同的一个宽度，否则不需要设置此属性的。

我把上面的程序稍作修改，使folder2 下面的条目拥有宽于窗口宽度的文字，则会得到如下效果：



（图 45）

可以发现，当 folder2 展开的时候，水平滚动条自动出现了，并且，当选中较短和较宽条目的时候，从条目背景色可以看出，它们拥有不同的宽度。如果我们调用 *tree.setFixedCellWidth(200);*设死所有条目宽度都为 250，那么将会是如下情况：



（图 46）

可见两部分的条目，不管它们各自的文字内容是多长，它们的长度都相同。

对于树组件来说，数据模型是比较重要的一点。AsWing 自带的 TreeModel 实现只有

DefaultTreeModel，它管理着由树状关系联系着的 TreeNode（MutableTreeNode 是可更改的 TreeNode 定义，DefaultMutableTreeNode 是它的默认实现类），TreePath 代表一个节点路径。本例中我们通过构造两层的 DefaultMutableTreeNode，形成了包含根结点在内的 3 层树结构，如果不希望显示根结点，可以调用 setRootVisible 方法进行设置。
这里我们需要重点了解一下 TreePath，对于 JList 来说，由于其数据结构的简单性，只需要通过条目的下标即可得到条目的值（或设置选中条目），因此上一节我们并没有介绍这一内容。而对于 JTree 来说，要表示一个选择，需要知道从根节点到子节点的一个序列，TreePath 类正是为此而诞生。我们为上例程序加入如下代码：

```
        tree.addSelectionListener(__selectionChanged);
    }

    private function __selectionChanged(e:Event):void{
        trace("_______selection_______");
        var paths:Array = tree.getSelectionPaths();
        if(paths){
            for each(var path:TreePath in paths){
                trace(path.getLastPathComponent());
            }
        }
    }
```

对于图 46 的选择情况，我们将会得到如下输出：

```
_______selection_______
item one 9
this is a long item text, it's really very very long 0
item one 8
this is a long item text, it's really very very long 1
```

可见，*TreePath.getLastPathComponent()*方法应该是最有用的方法，且慢，在我们构造 DefaultTreeModel 的时候，使用的是 DefaultMutableTreeNode，这里的输出虽然是文本字符串，但实际上 *getLastPathComponent()*返回的值是 DefaultMutableTreeNode 对象，之所以这里我们得到的输出是纯净的字符串内容，这是因为 DefaultMutableTreeNode 的 toString 方法返回它所包含的值的字符串形式，这也正好说明了 JTree 条目上显示的字符串的来由。读者可以尝试修改上例中的 createItem 方法，返回非 String 类型的对象，看看会发生什么现象。

如果需要用程序设置选中一个条目，那么你必须构造此条目对应的 TreePath 对象，或者，调用 setSelectionRow 指定选中第几行（此方法需要关注树的展开情况，而构造 TreePath 对象则不需要）。

对于自定义 JTree 的单元格，方法和 JList/JTable 类似，不同的地方是，JTree 有一个习惯，就是对于叶子节点和非叶子节点，通常需要不同形状的图标，并且在 LookAndFeel 中对此图标也有定义，但是单元格并不强求必须要有图标，默认的单元格实现 DefaultTreeCell 含有图标支持，你也可以实现一个完全没有图标的单元格。

这里我们用一个比较常用的例子来介绍，很多需求都要求叶子节点根据不同的值拥有不同的图标，比如 IM 的好友列表，条目的图标可能是根据好友的性别不同而使用不同的图标，也可能每个好友的图标都完全不同。我们以上例程序的模型为基础，当单元格值的最后一个字符数字为奇数时，使用圆形图标，偶数时，使用方形图标，实现如下单元格类：

MyTreeCell.as

```
package{

import org.aswing.Icon;
import org.aswing.tree.DefaultTreeCell;
import org.aswing.tree.MutableTreeNode;

public class MyTreeCell extends DefaultTreeCell{

    private var squareIcon:Icon;
    private var dotIcon:Icon;

    public function MyTreeCell(){
        super();
        squareIcon = new SquareIcon();
        dotIcon = new DotIcon();
    }

    override public function setCellValue(value:*) : void {
        super.setCellValue(value);
        //根据此Cell的值，如果最后一个字符数字是偶数，则使用方形图标
        //否则使用圆形图标
        var node:MutableTreeNode = MutableTreeNode(value);
        var str:String = node.getUserObject();
        var number:int = parseInt(str.charAt(str.length-1));
        if(number % 2 == 0){
            leaf_icon = squareIcon;
        }else{
            leaf_icon = dotIcon;
        }
    }
}
}

import org.aswing.Icon;
import org.aswing.graphics.Graphics2D;
import org.aswing.Component;
import org.aswing.graphics.SolidBrush;
import org.aswing.ASColor;
import flash.display.DisplayObject;
```

```
class SquareIcon implements Icon{

    public function SquareIcon(){}

    public function getIconWidth(c:Component):int{
        return 12;
    }

    public function getIconHeight(c:Component):int{
        return 12;
    }

    public function updateIcon(c:Component, g:Graphics2D, x:int,
y:int):void{
        g.fillRectangle(new SolidBrush(ASColor.GREEN), x, y, 12, 12);
    }

    public function getDisplay(c:Component):DisplayObject{
        return null;
    }
}


class DotIcon implements Icon{

    public function DotIcon(){}

    public function getIconWidth(c:Component):int{
        return 12;
    }

    public function getIconHeight(c:Component):int{
        return 12;
    }

    public function updateIcon(c:Component, g:Graphics2D, x:int,
y:int):void{
        g.fillCircle(new SolidBrush(ASColor.RED), x+6, y+6, 6);
    }

    public function getDisplay(c:Component):DisplayObject{
        return null;
```

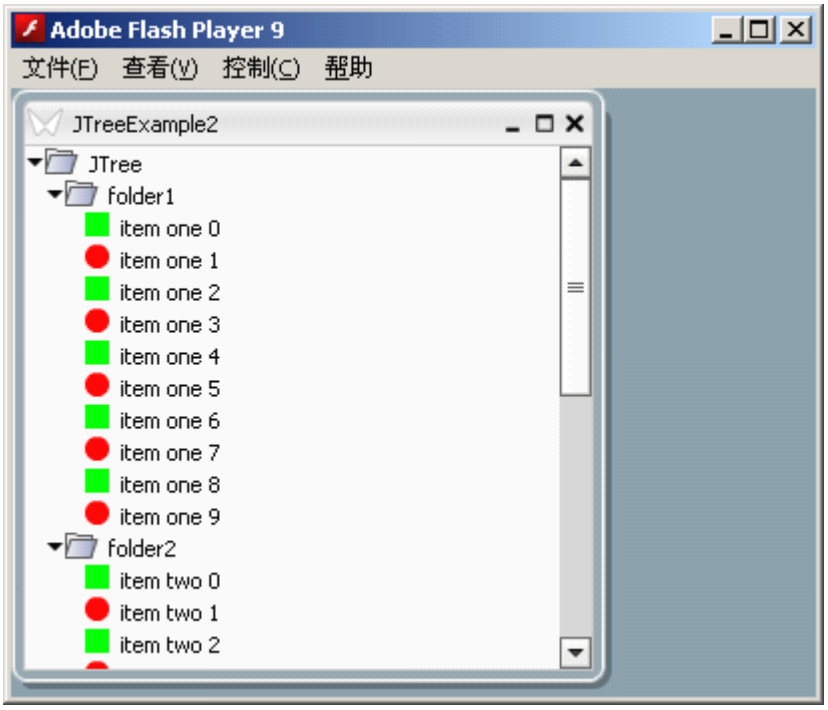
```
    }
}
```

我们继承自 DefaultTreeCell，是因为 DefaultTreeCell 已经拥有了图标+文字的能力，只需要在设置单元格值的时候，改变叶子图标成员变量即可，此类的运作原理，希望读者结合阅读 DefaultTreeCell 类源代码来完全理解。

而主程序，只需要在 JTreeExample1 的基础上在加入如下一句代码即可：

```
        tree.setCellFactory(new GeneralTreeCellFactory(MyTreeCell));
```

整个类代码这里就不贴出来了，编译运行，可得到如下效果：



（图 47）

可见，叶子节点的图标达到了预期的效果。这里再次提醒，JTree 并不要求单元格必须支持图标，这完全由单元格自己来决定，你也可以实现一个没有图标的单元格，也可以实现用任何组件来做单元格，就像 JTable/JList 的单元格那样，开发者拥有完全的自主权。

树的节点也支持编辑器，就像 JTable 的 TableCellEditor 那样，它们两者拥有共同的父接口 CellEditor，实现方法几乎完全一样，需要使用此功能的读者，可以参阅 2.5.4 节 —— 快速修改时长和状态（CellEditor）。

另外，要掌握 JTree 的所有功能，还需要了解众多方法，比如节点的展开，收缩，设定是否自动滚动展开位置，这些都有相应的接口方法可以调用，具体请参阅 api 文档。

2.7.4 标签面板（JTabbedPane，JAccordion，JClosableTabbedPane）

标签面板在需要分页显示面板的应用程序中极为常用，比如 IE 浏览器的选项设置，

Firefox 浏览器的标签页，都是标签面板的典型应用。

AsWing 提供的标签面板 JTabbedPane 拥有类似 IE 浏览器的选项卡那样的功能，而 JClosableTabbedPane 则是在 JTabbedPane 的基础上，增加了标签关闭按钮，从而保证 Firefox 那样的标签页能够得以实现。

AsWing 中的 JTabbedPane 被实现为一个拥有标签的容器，因此你可以把它当作普通的容器来使用，只不过它比空容器多了标签页，通过用户点击标签页，可以切换内容组件的显示。下面我们通过一个例子来尽量展示 JTabbedPane 的所有功能：

JTabbedPaneExample.as

```
package{

import flash.display.Sprite;
import flash.events.*;

import org.aswing.*;
import org.aswing.event.InteractiveEvent;

public class JTabbedPaneExample extends Sprite{

    private var tabbedPane:JTabbedPane;
    private var statusLabel:JLabel;
    private var placementButton:JButton;
    private var frame:JFrame;

    public function JTabbedPaneExample(){
        AsWingManager.initAsStandard(this);

        frame = new JFrame(null, "JTabbedPaneExample");

        tabbedPane = new JTabbedPane();

        var panel:JPanel = new JPanel();
        var addTabButton:JButton = new JButton("add tab");
        var removeTabButton:JButton = new JButton("remove tab");
        removeTabButton.setToolTipText("Remove a random tab");
        panel.append(new JButton("button2"));
        panel.append(new JButton("button3"));
        //添加第1个Tab
        tabbedPane.appendTab(
            panel,
            "Buttons",
            new ColorIcon(ASColor.GREEN, 30, 20), "tip for tab");
```

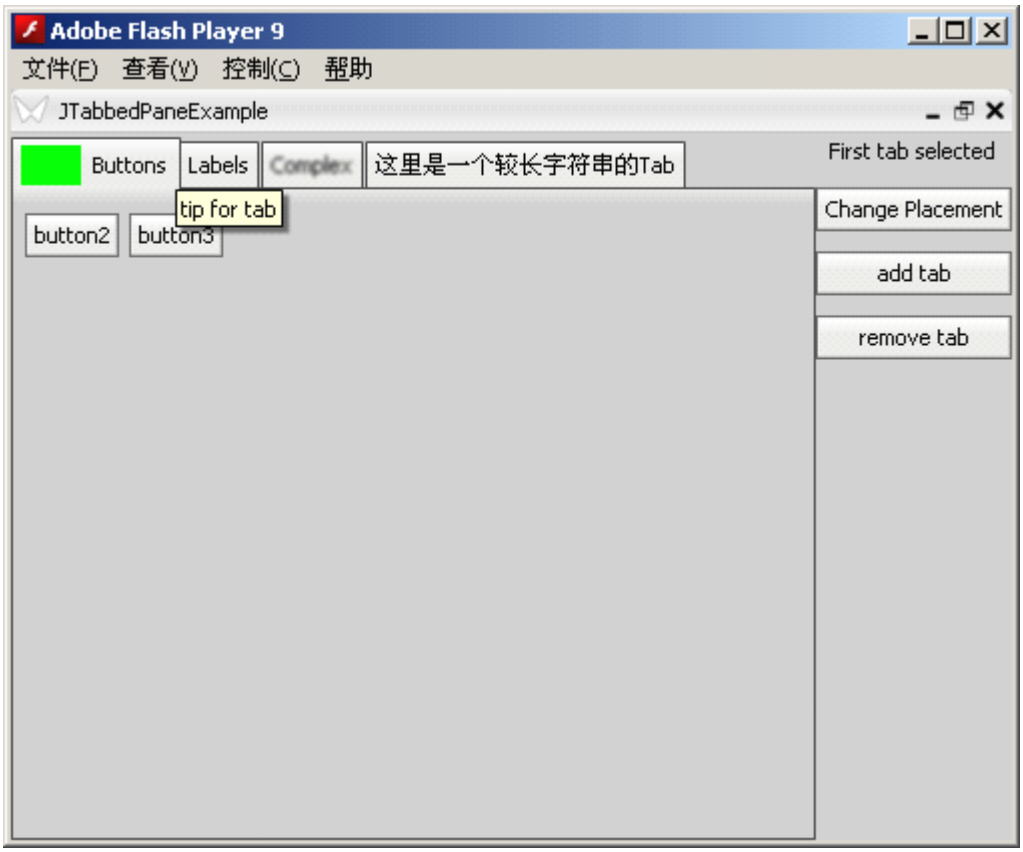
```
        panel = new JPanel();
        panel.append(new JLabel("label1"));
        panel.append(new JLabel("label2"));
        panel.append(new JLabel("label3"));
        //添加第2个Tab
        tabbedPane.appendTab(panel, "Labels", null, "tip2\nline2");

        panel = new JPanel(new BorderLayout());
        panel.append(new JTextField("JTextField"),
            BorderLayout.NORTH);
        var p:JPanel = new JPanel();
        p.append(new JCheckBox("JCheckBox"));
        p.append(new JRadioButton("JRadioButton"));
        panel.append(p, BorderLayout.CENTER);
        //添加第3个Tab
        tabbedPane.appendTab(panel, "Complex", null, "Disabled");

        frame.getContentPane().append(tabbedPane,
            BorderLayout.CENTER);

        //右边的工具栏
        var rightPane:Container = SoftBox.createVerticalBox(10);
        statusLabel = new JLabel("First tab selected");
        placementButton = new JButton("Change Placement");
        rightPane.append(statusLabel);
        rightPane.append(placementButton);
        rightPane.append(addTabButton);
        rightPane.append(removeTabButton);
        frame.getContentPane().append(rightPane,
            BorderLayout.EAST);

        //添加第4个Tab
        tabbedPane.appendTab(
            new JButton("Button"),
            "这里是一个较长字符串的Tab");

        tabbedPane.addEventListener(
            InteractiveEvent.STATE_CHANGED,
            __selectionChanged);
        placementButton.addActionListener(__changePlacement);
        addTabButton.addActionListener(__addTab);
        removeTabButton.addActionListener(__removeTab);
        //设置第3个Tab为无效状态
        tabbedPane.setEnabledAt(2, false);
```

```
        frame.setSizeWH(400, 380);
        frame.show();
    }

    //运行时添加一个Tab
    private function __addTab(e:Event):void{
        var p:JPanel = new JPanel(new BorderLayout());
        p.append(
            new JLabel("下面是一个JTextArea"),
            BorderLayout.NORTH);
        p.append(
            new JScrollPane(new JTextArea("", 10, 10)),
            BorderLayout.CENTER);
        var n:int = tabbedPane.getComponentCount();
        tabbedPane.insertTab(
            Math.floor(Math.random()*n),
            p, "新添加");
    }
    //运行时随机移除一个Tab
    private function __removeTab(e:Event):void{
        var n:int = tabbedPane.getComponentCount();
        tabbedPane.removeTabAt(
            Math.floor(Math.random()*n));
    }
    //Tab选择改变时
    private function __selectionChanged(e:Event):void{
        var index:Number = tabbedPane.getSelectedIndex();
        statusLabel.setText("Selected : " + index);
    }
    //运行时改变Tab方位
    private function __changePlacement(e:Event):void{
        var placement:int = tabbedPane.getTabPlacement();
        if(placement == JTabbedPane.LEFT){
            tabbedPane.setTabPlacement(JTabbedPane.RIGHT);
        }else if(placement == JTabbedPane.RIGHT){
            tabbedPane.setTabPlacement(JTabbedPane.BOTTOM);
        }else if(placement == JTabbedPane.BOTTOM){
            tabbedPane.setTabPlacement(JTabbedPane.TOP);
        }else{
            tabbedPane.setTabPlacement(JTabbedPane.LEFT);
        }
    }
}
```

```
}

import org.aswing.graphics.Graphics2D;
import org.aswing.Icon;
import org.aswing.Component;
import flash.display.DisplayObject;
import org.aswing.ASColor;
import org.aswing.graphics.SolidBrush;
import flash.display.Shape;

class ColorIcon implements Icon{

    private var color:ASColor;
    private var width:int;
    private var height:int;
    private var shape:Shape;

    public function ColorIcon(color:ASColor, width:int, height:int){
        shape = new Shape();
        this.color = color;
        this.width = width;
        this.height = height;
    }

    public function updateIcon(
        com:Component, g:Graphics2D, x:int, y:int):void{
        shape.graphics.clear();
        g = new Graphics2D(shape.graphics);
        g.fillRectangle(new SolidBrush(color),
            x, y, width, height);
    }

    public function getIconHeight(com:Component):int{
        return height;
    }

    public function getIconWidth(com:Component):int{
        return width;
    }

    public function getDisplay(com:Component):DisplayObject{
        return shape;
    }
```

```
}
```

编译并运行此程序，把窗口最大化后，我们能得到如下界面：



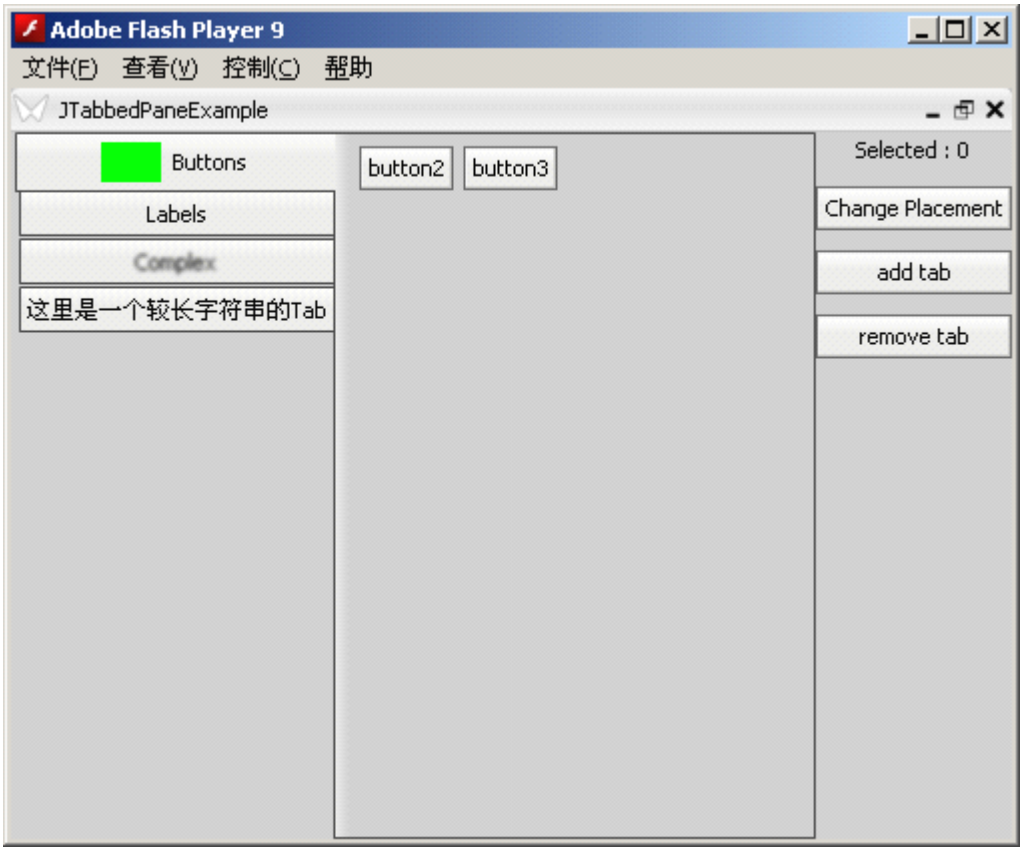
（图 48）

我们看到了一个类似 IE 浏览器选项卡的界面，并且第一个标签（Tab）还有一个图标，当鼠标移到第一个标签上停留片刻，可以看到"tip for tab"字样的工具提示，第三个标签显示为模糊状，点击它没有任何反应，点击其他标签，则会改变标签的选中状态，并同时下部显示出对应的组件。这正是一个标签面板的正确行为。我们看看添加一个标签的方法定义：

```
public function appendTab(com:Component, title:String="",
icon:Icon=null, tip:String=null):void
```

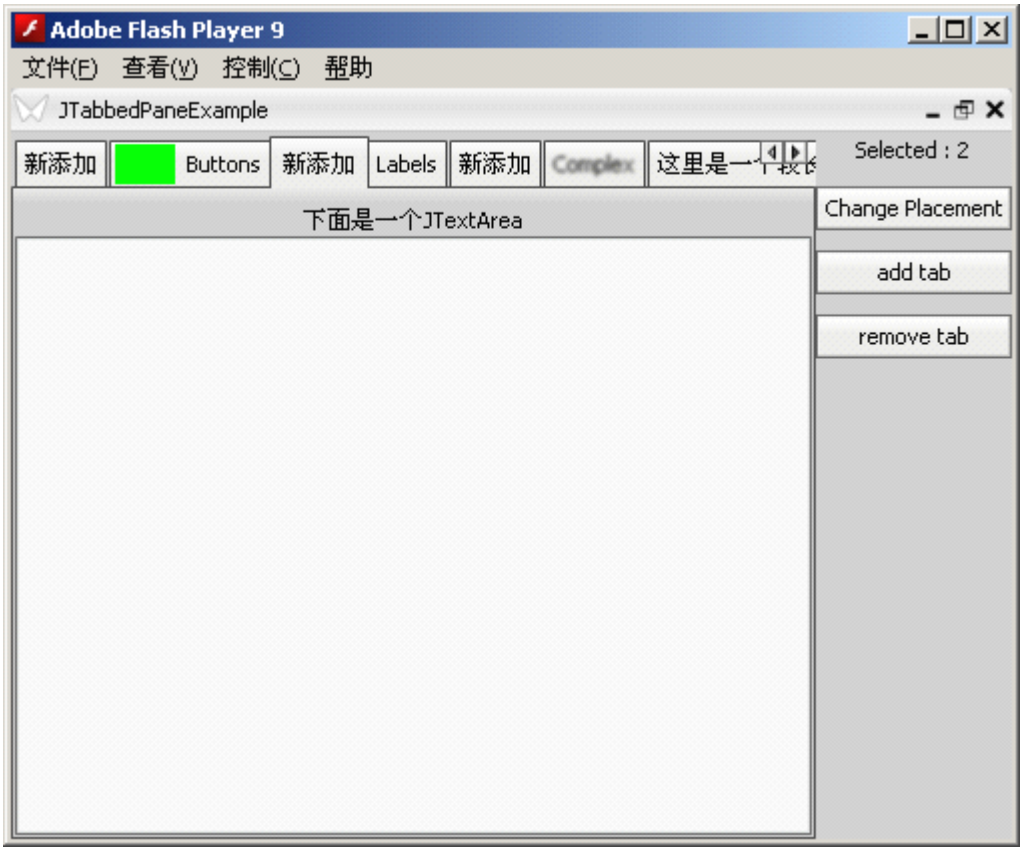
第一个参数是要添加到标签面板容器中的组件，第二个参数指定标签文本，第三个参数指定标签的图标（null 代表不需要图标），最后一个参数指定标签的工具提示（null 代表不需要提示）。读者可以参阅上例中的代码结合理解此方法的使用。

setEnabledAt 方法可以设置指定标签是否激活，非激活的标签，用户点击后无任何反应。另外，还可以通过 setTabPlacement 方法改变标签所处的方位，点击"Change Placement"按钮后，我们将得到如下界面：



（图 49）

标签的方位可以被设置为上，下，左，右，分别对应的值为 *JTabbedPane.TOP*，*JTabbedPane.BOTTOM*，*JTabbedPane.LEFT*，*JTabbedPane.RIGHT* 四个常量。作为容器，标签面板当然可以运行时动态添加和移除自组件，读者可以尝试点击"add tab"和"remove tab"试看效果。当被添加的标签过多，当前界面显示不全时，JTabbedPane 会自动出现左右移动按钮，可使标签左右滚动，以显示出原本在外的标签项，如下图：



（图 50）

当然，你也可以在运行时更改标签文字或图标，甚至提示文字，对应的方法分别为 setTitleAt, setIconAt 和 setTipAt。详细信息请参阅 AbstractTabbedPane 类（JTabbedPane 的直接父类）的 api 文档。

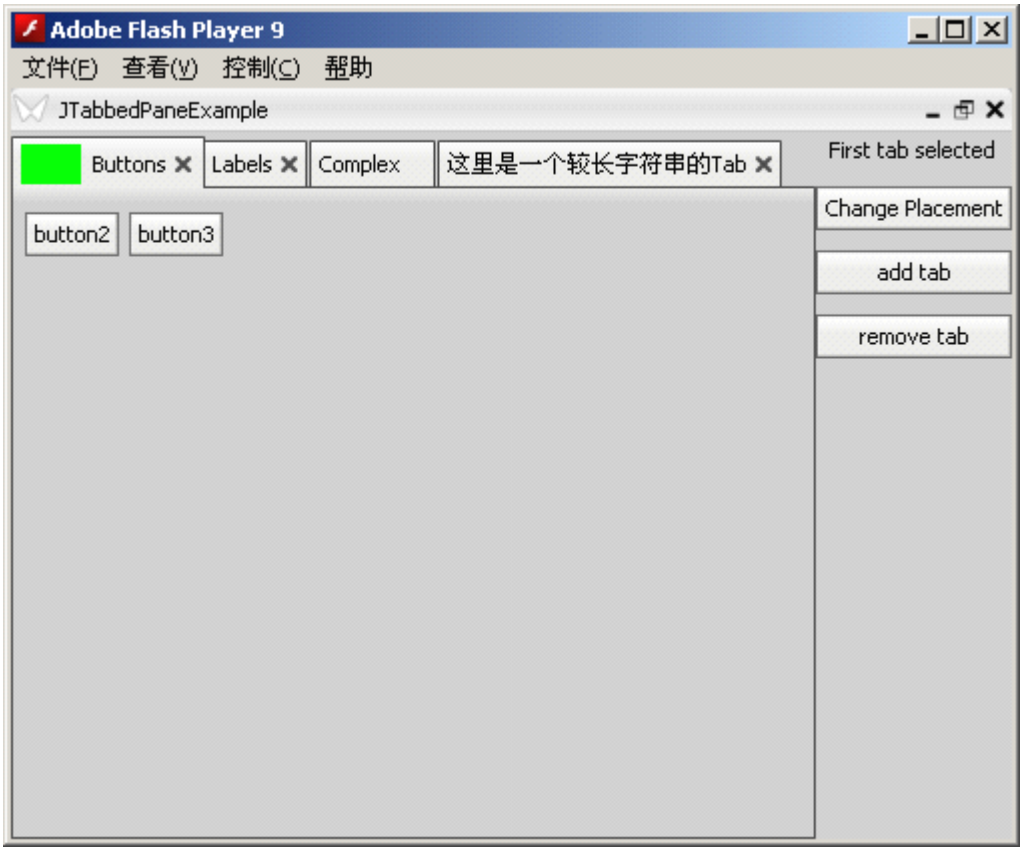
对于含有关闭按钮的标签面板 JClosableTabbedPane，它是 JTabbedPane 的子类，增加了如下两个方法：

```
public function setCloseEnabledAt(index:int,
enabled:Boolean):void

public function isCloseEnabledAt(index:int):Boolean
```

以支持指定标签是否可通过关闭按钮关闭，默认值都为 true（可关闭）。如果我们把上例程序的第 21 行改为：

```
        tabbedPane = new JClosableTabbedPane();
```

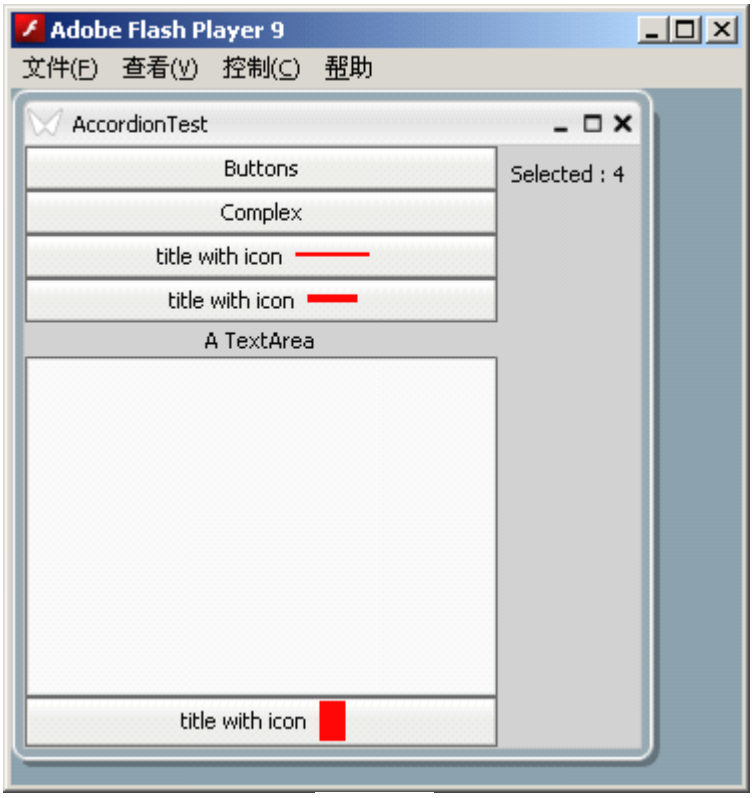
得到的程序界面则会像下图：



（图 51）

第三项由于是非激活状态，不能有效点击，当然也就不能被关闭，所以关闭按钮并没有出现。

AsWing 还含有另一个功能类似的容器组件折叠窗 JAccordion，他与 JTabbedPane 同样继承自 AbstractTabbedPane 类，用户几乎同样的方法，只是表现形式不同而已，这里就不再多述，下面给出一张截图，以便读者了解此组件的表现形式：



（图 52）

提示：AbstractTabbedPane 类的 setVisibleAt 方法，并不被 JTabbedPane 支持，调用它会抛出异常，而 JAccordian 支持此方法用来隐藏指定的页。

2.7.5 滑动条,进度条和滚动条（JSlider，JProgressBar 和 JScrollBar）

滑动条，进度条和滚动条（JSlider，JProgressBar 和 JScrollBar）是三个具有诸多共同点的组件，也是常用的组件。它们拥有共同的模型类型——BoundedRangeModel，都拥有两种朝向——水平（HORIZONTAL）或者竖直（VERTICAL）。但是，它们之间通常并不能互相取代，因为它们的操作和表现方式决定了它们将被应用到不同的功能上，下例将通过一个简单的滑动条功能来展示它们之间的异同。

见此程序：

JSliderExample.as

```
package{

import flash.display.DisplayObject;
import flash.display.Sprite;
import flash.events.Event;
```

```
import org.aswing.AsWingManager;
import org.aswing.AssetPane;
import org.aswing.BorderLayout;
import org.aswing.BoundedRangeModel;
import org.aswing.DefaultBoundedRangeModel;
import org.aswing.JFrame;
import org.aswing.JPanel;
import org.aswing.JProgressBar;
import org.aswing.JScrollBar;
import org.aswing.JScrollPane;
import org.aswing.JSlider;
import org.aswing.SoftBoxLayout;
import org.aswing.border.TitledBorder;

public class JSliderExample extends Sprite{

    private var slider:JSlider;
    private var scrollbar:JScrollBar;
    private var progressbar:JProgressBar;
    private var model:BoundedRangeModel;
    private var assetPane:AssetPane;

    [Embed(source="paint1.jpg")]
    private var imgClass:Class;

    public function JSliderExample(){
        super();
        AsWingManager.initAsStandard(this);

        var asset:DisplayObject = DisplayObject(new imgClass());
        assetPane = new AssetPane(asset, AssetPane.PREFER_SIZE_IMAGE);

        var frame:JFrame = new JFrame(this, "JSliderExample");
        var pane:JPanel = new JPanel(new BorderLayout());
        pane.append(new JScrollPane(assetPane));
        var bottom:JPanel = new JPanel(new
SoftBoxLayout(SoftBoxLayout.Y_AXIS, 2));
        bottom.setBorder(new TitledBorder(
            null,
            "Bars",
            TitledBorder.TOP,
            TitledBorder.LEFT,
            12,
            6));
```

```
        pane.append(bottom, BorderLayout.SOUTH);

        //创建界限范围模型，默认值100，滑块占幅200，最小值0，最大值1000
        model = new DefaultBoundedRangeModel(100, 200, 0, 1000);
        //创建滑动条
        slider = new JSlider(JSlider.HORIZONTAL);
        slider.setModel(model);
        slider.setShowValueTip(true);
        slider.setPaintTicks(true);
        slider.setMajorTickSpacing(200);
        slider.setMinorTickSpacing(20);
        slider.setPaintTrack(true);
        //创建滚动条
        scrollbar = new JScrollBar(JScrollBar.HORIZONTAL);
        scrollbar.setModel(model);
        //创建进度条
        progressbar = new JProgressBar(JProgressBar.HORIZONTAL);
        progressbar.setModel(model);

        bottom.appendAll(slider);
        //先不加入滚动条和进度条
        //bottom.appendAll(slider, scrollbar, progressbar);

        model.addStateListener(__stateChanged);

        frame.setContentPane(pane);
        frame.setSizeWH(300, 300);
        frame.show();
    }

    private function __stateChanged(e:Event):void{
        var scale:Number = model.getValue();
        //用下面注释掉的语句，也能得到相同结果
        //var scale:Number = slider.getValue();
        assetPane.setCustomScale(scale);
        trace(scale);
    }
}
}
```

编译并运行，将得到如下界面：

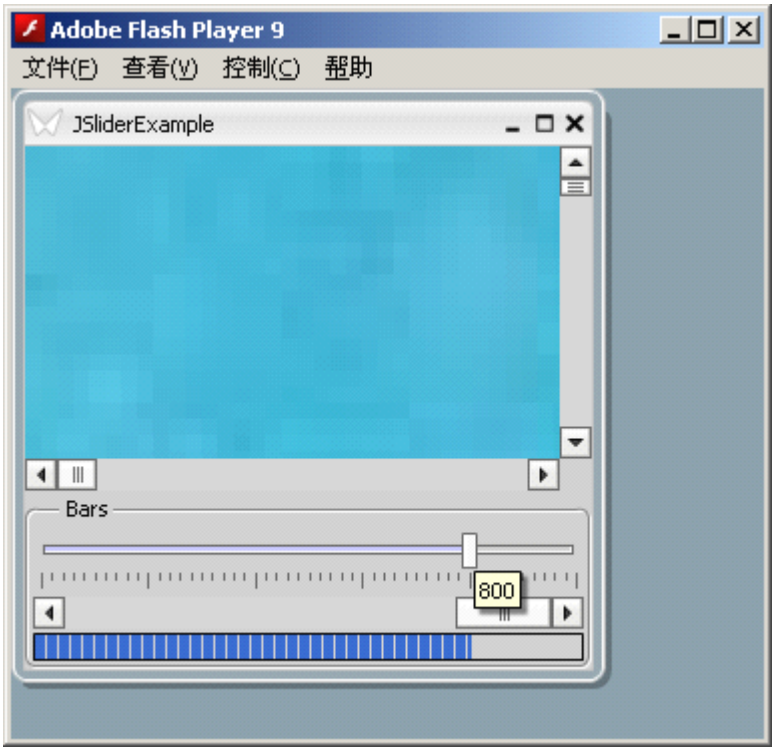
（图 53）

窗口下部的滑动条，控制了上面部分图片的缩放，滑动条模型的默认值我们设置了为 100，作用与图片的缩放就是 100%，如果把滑动条从左拖到右，模型的值会从 0 变化到 1000，能看到图片从 0 大小变化到 10 倍大。

这就是滑动条通常所做的事情——调节一个有限范围的数值。

默认情况下，滑动条是没有上图中的刻度的，也没有当前值的 ToolTip 显示等，这些都需要通过一些列方法来设置，具体请见上例程序中的相关代码，读者可以尝试改变它们的值来验证它们的功能。需要注意的是，刻度的显示需要设置两个值，大刻度间距（setMajorTickSpacing）和小刻度间距（setMinorTickSpacing），大刻度间距必须大于小刻度间距，否则刻度将无法绘制，并且，至少要保证小刻度间距不小于模型值范围，否则将无法绘制出一个完整的刻度。上例中，我们的模型值范围为 1000-0=1000，我们设置大刻度间距为 200，小刻度间距为 20，因此大刻度把整个范围划分成了 1000/200=5 份，小刻度则把范围划分成了 1000/20=50 份，图 53 正好是如此。

我们将上例代码里注释的代码部分替换为其上部的代码，把创建的滚动条和进度条也加入面板，运行效果将是如下：



（图 54）

下面部分也呈现出了滚动条和进度条，在这个程序中，不管拖动滚动条还是滑动条，都将得到同样的效果，图片会缩放，并且滑动条，滚动条和进度条会始终保持相同的进度，这是因为我们在程序中给他们设置了同一个模型，它们的数据完全一样，只是表现形式不同罢了。在这个例子中，我们也将看到一个特点，那就是滚动条能够拖动到最靠右边的位置，而滑动条和进度条都在 800 刻度处就不能再往右移了。这是因为我们创建 DefaultBoundedRangeModel 的时候，第二个参数滑块占幅为 200，因此模型能够得到的最大值即为 1000-0-200=800，因此滑动条和进度条都只能停靠在 800 刻度处，那为什么滚动条又能停靠在最靠右的地方呢——这即是滚动条与其他两个组件的不同之处，滚动条的滑块表现被设定会占用幅度，因此 800 刻度加 200 滑块等于 100 正是滚动条能到达的最右的位置，而滑动条和进度条，前者的滑块表现被设定为不占用幅度，后者根本就没有滑块，因此它们就表现为右边空余 200 刻度的情形。

通常情况下，使用滑动条和进度条时，滑块占幅都设置为 0，就不会出现上例的空余刻度情况。而滚动条，滑块幅度承担着表现显示区域的范围大小的作用，所以通常会设置一个合理的值，比如在滚动一段文本时，滑块占幅应该是文本框的高度，模型值的范围应该是文本字符的总高度。滚动面板（JScrollPane）里面的滚动条，很好的体现了这一特性。

2.7.5 其他常用组件

由于篇幅有限，剩下的 AsWing 自带组件不能一一进行详细讲解，这里列出所有以往章节未曾遇到过或者遇到过但是没有讲解的组件，进行一个简单的介绍。

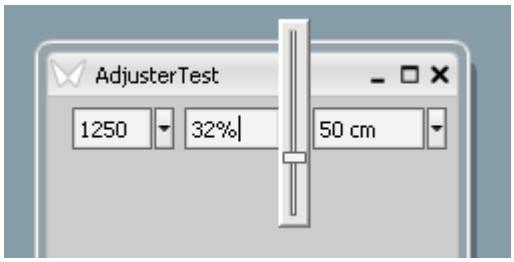
JRootPane 和 JPopup： JRootPane 是所有根容器（比如 JFrame，JWindow，JPopup 等）的基类，它内部封装了作为一个根容器的键盘管理（KeyboardManager）功能，任何继承自

它的容器，都能拥有独立的键盘管理功能，比如快捷助记符（Mnemonic），回车键触发的默认按钮等。JPopup 继承自 JRootPane，是一个最原始的弹出容器，它是 JWindow 的父类，它作为弹出根容器，最大的特点就是不用显式的被 addChild 或者 append 到舞台中，只要创建它的实例，然后调用 show（或者 *setVisible(true)*）方法即可让它显示到舞台上。从前面章节介绍过的 JWindow 和 JFrame 的使用方法即可看出这一点。

AssetPane(JLoadPane, JAttachPane)：资源面板，这里的资源是指显示元件（DisplayObject），前面章节我们讲到过可以通过 addChild 添加任意显示原件到 AsWing 组件中，但是如果你想让所加入的显示原件也能像组件一样的被布局管理器自动布局，那么可以使用 AssetPane 来包装这个显示原件。JLoadPane 和 JAttachPane 都是 AssetPane 的子类。

JLabelButton：标签按钮，表现形式就像一个纯文本超级链接，它是 AbstractButton 的子类，因此拥有按钮的所有功能。

JAdjuster：调节器，用于让用户选择一个范围内的值，也使用 BoundedRangeModel，表现形式为一个输入框，右边一个下拉按钮，点击后会弹出一个滑动条（Slider）。如图 55:



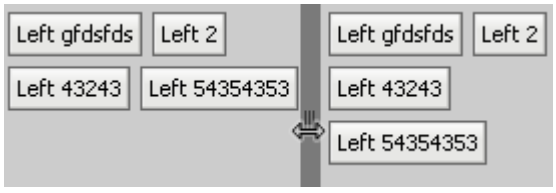
（图 55）

JPopupMenu：弹出菜单，它是一个菜单容器，使用它可以在任何地方弹出一个菜单面板，JMenu 实际上是使用了一个 JPopupMenu 来作为子菜单的容器。

JSeparator：分隔条，纯界面组件，无实际功能，以一个凹槽条的形式来美化界面需要的间隔。

JSpacer：占位器，无实际功能，通常也无界面表现（全透明），用来在布局管理器中进行占位，也可以起到分割界面的作用。

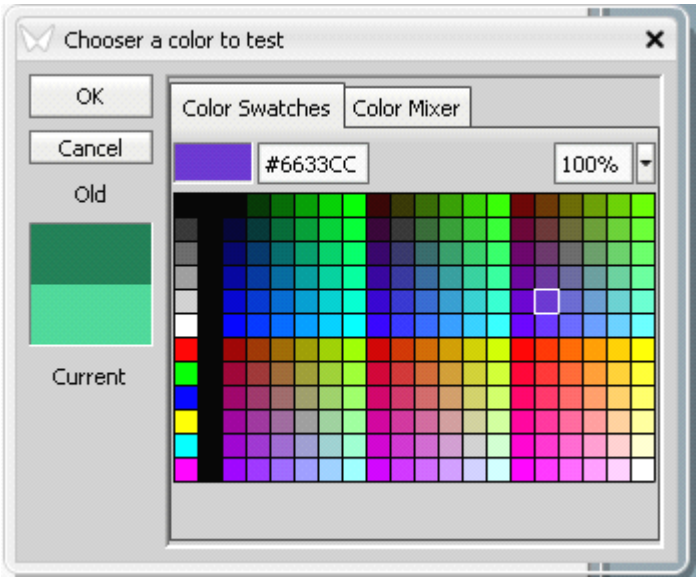
JSplitPane：分割面板，此容器可以容纳两个组件，使其左右/上下分布，中间会有一个分隔条，并且用户可以拖动分隔条进行左右/上下空间分布的调节。如图 56：



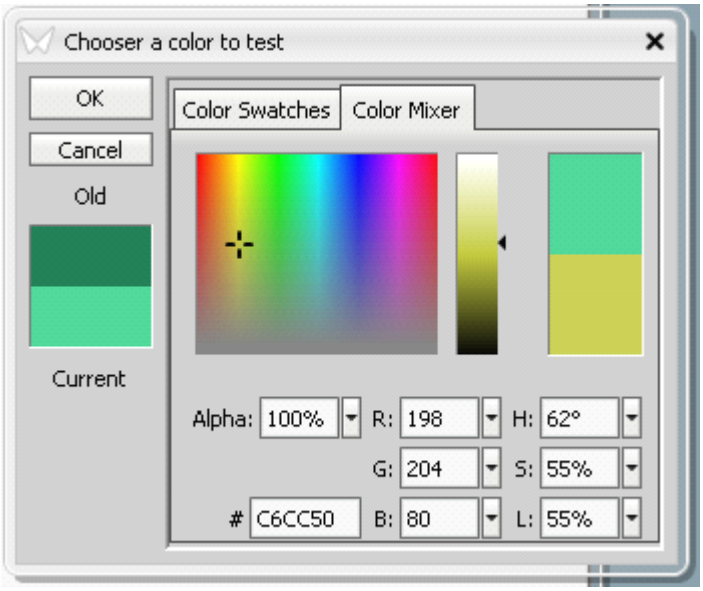
（图 56）

JColorChooser(JColorMixer,JColorSwatches)：颜色选择器，JColorChooser 由两部分组成，混色器（JColorMixer）和样色板（JColorSwatches），并且它们也是可以单独使用

的组件。如图 57，58：



（图 57）



（图 58）

Folder：纸夹，它是一个拥有标题条的容器，并且点击标题条可以展开或收缩内容面板。

GridList：格子列表，它是列表组件表现形式的扩展，拥有多行多列，并且也可以自定义单元格。可以用于按行排列多个列的内容，并且支持自动换行的效果，而且，它实现了 Viewportable 接口，可以被 JScrollPane 友好包装。

2.8 创建自己的组件(数字步进器 NumericStepper)

一个 UI 库自带的组件常常不能完全解决我们所有的需求，虽然大多数功能界面都可以用现有组件搭配而成，但是有些时候，还是需要自己来创建一些组件，以满足自己独特的需求。

有的 UI 库是含有数字步进器 NumericStepper 的，但是 AsWing 没有附带这一组件，AsWing 提供了另外一个组件调节器 JAdjuster，希望以之满足大多数数字调节的需求。然而，有些地方你仍然希望调节数字的组件是 NumericStepper 那样的，这个时候，你就只能自己动手编写一个 NumericStepper。下面我们将以编写 NumericStepper 为例，介绍如何创建自己的组件。

NumericStepper 的表现形式为左边一个输入框，右边上下排列着两个箭头按钮。我们很容易就能想到，采用 BorderLayout+BoxLayout 布局一个文本框加两个按钮即可达成。于是我们首先编写 NumericStepper 的表现形式部分的代码，如下：

NumericStepper.as

```
package{

import org.aswing.ASColor;
import org.aswing.BorderLayout;
import org.aswing.Box;
import org.aswing.EditableComponent;
import org.aswing.Insets;
import org.aswing.JButton;
import org.aswing.JPanel;
import org.aswing.JTextField;

public class NumericStepper extends JPanel{

    protected var textField:JTextField;
    protected var upButton:JButton;
    protected var downButton:JButton;

    public function NumericStepper(columns:int=3){
        super(new BorderLayout());
        textField = new JTextField("", columns);
        upButton = new JButton(null,
            new ArrowIcon(
                -Math.PI/2,
                8,
                new ASColor(0x444444),
```

```
                new ASColor(0x0)));
        downButton = new JButton(null,
            new ArrowIcon(
                Math.PI/2,
                8,
                new ASColor(0x444444),
                new ASColor(0x0)));
        append(textField, BorderLayout.CENTER);
        var right:Box = Box.createVerticalBox();
        right.appendAll(upButton, downButton);
        append(right, BorderLayout.EAST);

        upButton.setMargin(new Insets(0, 3, 0, 3));
        downButton.setMargin(new Insets(0, 3, 0, 3));
        upButton.setFocusable(false);
        downButton.setFocusable(false);
    }

}
}

import org.aswing.graphics.*;
import org.aswing.*;
import org.aswing.geom.*;
import flash.display.DisplayObject;
import org.aswing.plaf.UIResource;
import flash.geom.Point;

class ArrowIcon implements Icon, UIResource{

    private var arrow:Number;
    private var width:int;
    private var height:int;
    private var shadow:ASColor;
    private var darkShadow:ASColor;

    public function ArrowIcon(arrow:Number, size:int, shadow:ASColor,
            darkShadow:ASColor){
        this.arrow = arrow;
        this.width = size;
        this.height = size;
        this.shadow = shadow;
        this.darkShadow = darkShadow;
    }
```

```
    public function updateIcon(c:Component, g:Graphics2D, x:int,
y:int):void{
        var center:Point = new Point(c.getWidth()/2, c.getHeight()/2);
        var w:Number = width;
        var ps1:Array = new Array();
        ps1.push(nextPoint(center, arrow, w/2/2));
        var back:Point = nextPoint(center, arrow + Math.PI, w/2/2);
        ps1.push(nextPoint(back, arrow - Math.PI/2, w/2));
        ps1.push(nextPoint(back, arrow + Math.PI/2, w/2));

        var ps2:Array = new Array();
        ps2.push(nextPoint(center, arrow, w/2/2-1));
        back = nextPoint(center, arrow + Math.PI, w/2/2-1);
        ps2.push(nextPoint(back, arrow - Math.PI/2, w/2-2));
        ps2.push(nextPoint(back, arrow + Math.PI/2, w/2-2));

        g.fillPolygon(new SolidBrush(darkShadow), ps1);
        g.fillPolygon(new SolidBrush(shadow), ps2);
    }

    protected function nextPoint(p:Point, dir:Number,
dis:Number):Point{
        return new Point(p.x+Math.cos(dir)*dis, p.y+Math.sin(dir)*dis);
    }

    public function getIconHeight(c:Component):int{
        return width;
    }

    public function getIconWidth(c:Component):int{
        return height;
    }

    public function getDisplay(c:Component):DisplayObject{
        return null;
    }

}
```

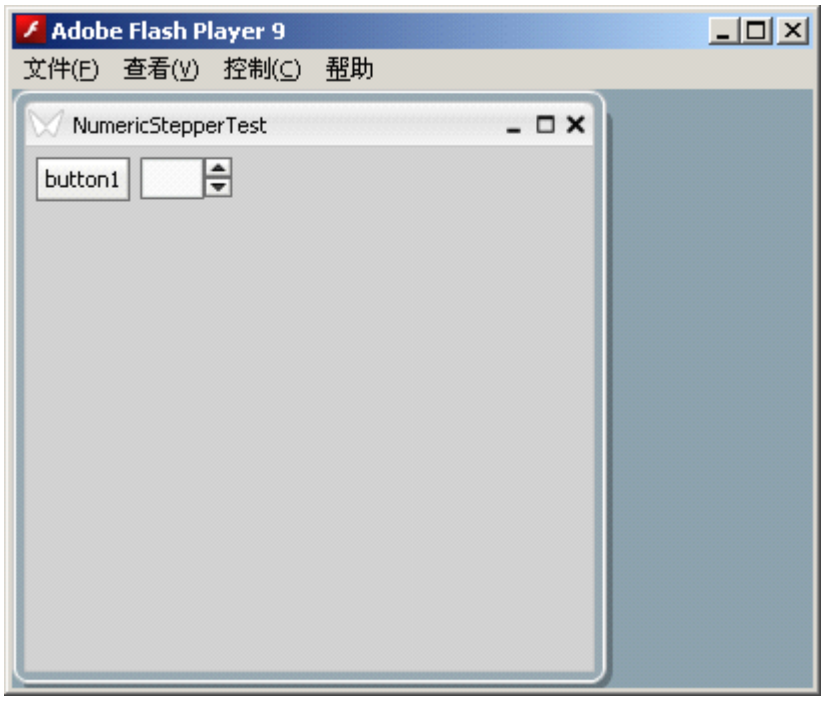
我们让 NumericStepper 继承自 JPanel，然后创建了一个 JTextField，两个 JButton，重点实现了表现箭头的图标 ArrowIcon，可见除了 ArrowIcon 实现上有些复杂之外，其他部分都非常简单（当然如果你拥有一个好的美术设计师，你可以直接使用 AssetIcon 绑定设计师创建的箭头图案，则更为方便）。为了测试我们这个组件的外观，我们创建一个测试程序

来显示他，程序如下：

NumericStepperTest.as

```
package{

import flash.display.Sprite;

import org.aswing.AsWingManager;
import org.aswing.JButton;
import org.aswing.JFrame;
import org.aswing.JPanel;

public class NumericStepperTest extends Sprite{

    public function NumericStepperTest(){
        super();
        AsWingManager.initAsStandard(this);
        var frame:JFrame = new JFrame(null, "NumericStepperTest");
        var pane:JPanel = new JPanel();
        var ns:NumericStepper = new NumericStepper();
        pane.appendAll(new JButton("button1"), ns);
        frame.setContentPane(pane);
        frame.setSizeWH(300, 300);
        frame.show();
    }

}
}
```

编译并运行，将看到如下画面：



（图 59）

为了对比，我们在它之前加入了一个普通按钮，从图示可见，我们的 NumericStepper 已经像模像样了，只是文本输入框的边框似乎如果能够扩展到箭头按钮的右边，完全包裹住整个 NumericStepper 就更好了。为此，我们只需要去掉文本框的边框和背景色，然后给 NumericStepper 创建一个类似的边框和设置背景色，即可到达此效果，于是我们编写 NumericStepperBorder 代码如下：

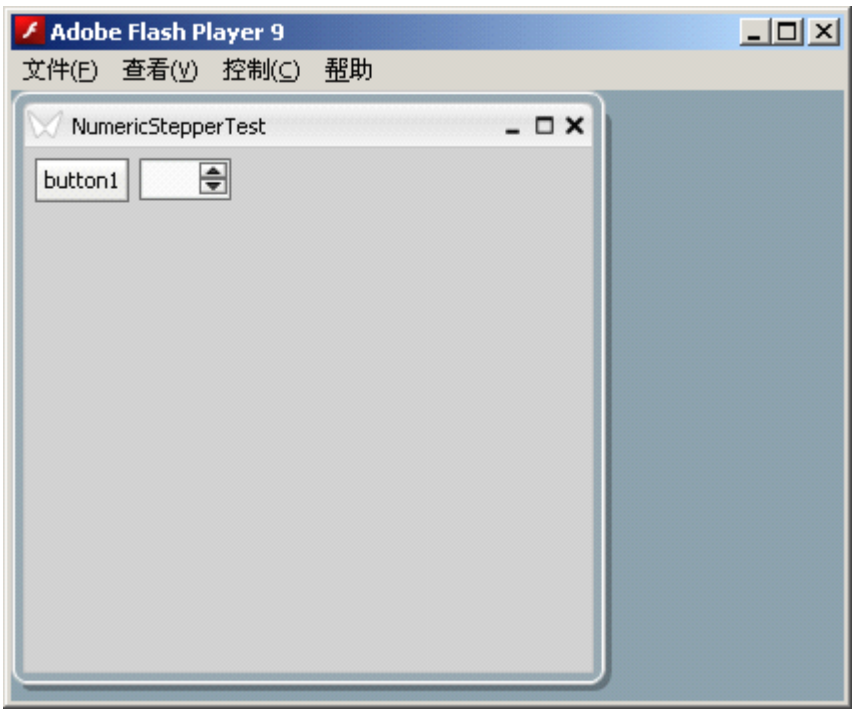
```
class NumericStepperBorder implements Border{

    private var light:ASColor;
    private var shadow:ASColor;

    public function NumericStepperBorder(){
        light = new ASColor(0xDCDEDD);
        shadow = new ASColor(0x666666);
    }

    public function updateBorder(c:Component, g:Graphics2D,
r:IntRectangle):void{
        var x1:Number = r.x;
        var y1:Number = r.y;
        var w:Number = r.width;
        var h:Number = r.height;
        var editable:Boolean = true;
        //TODO check is the NumericStepper is editable
        if(editable){
            g.drawRectangle(new Pen(shadow, 1), x1+0.5, y1+0.5, w-1,
h-1);
```

```
        }
        g.drawRectangle(new Pen(light, 1), x1+1.5, y1+1.5, w-3, h-3);

    }

    public function getBorderInsets(com:Component,
bounds:IntRectangle):Insets{
        return new Insets(2, 2, 2, 2);
    }

    public function getDisplay(c:Component):DisplayObject{
        return null;
    }
}
```

然后在 NumericStepper 的构造函数中，去除文本框的所有装饰，然后给 NumericStepper 加上此边框和白色背景色，代码如下：

```
        textField.setBorder(null);
        textField.setOpaque(false);
        textField.setBackgroundDecorator(null);

        setBorder(new NumericStepperBorder());
        setBackground(new ASColor(0xF3F3F3));
        setOpaque(true);
```

再次编译并运行测试程序，将得到如下画面：



（图 60）

可见画面效果已经完全达到我们预想的要求了，下面就是给它加上数据处理的能力了。在数据模型上，我们可以采用 JAdjuster 使用的 BoundedRangeModel，参考 JAdjuster 的源

代码，不难以给我们的 NumericStepper 加上数据模型，这里给出改动后的 NumericStepper 类代码：

```
public class NumericStepper extends JPanel{

    private var model:BoundedRangeModel;
    protected var textField:JTextField;
    protected var upButton:JButton;
    protected var downButton:JButton;
    protected var editable:Boolean;

    public function NumericStepper(columns:int=3){
        super(new BorderLayout());
        editable = true;
        textField = new JTextField("", columns);
        textField.setEditable(editable);
        textField.setBorder(null);
        textField.setOpaque(false);
        textField.setBackgroundDecorator(null);

        setBorder(new NumericStepperBorder());
        setBackground(new ASColor(0xF3F3F3));
        setOpaque(true);

        upButton = new JButton(null,
            new ArrowIcon(
                -Math.PI/2,
                8,
                new ASColor(0x444444),
                new ASColor(0x0)));
        downButton = new JButton(null,
            new ArrowIcon(
                Math.PI/2,
                8,
                new ASColor(0x444444),
                new ASColor(0x0)));
        append(textField, BorderLayout.CENTER);
        var right:Box = Box.createVerticalBox();
        right.appendAll(upButton, downButton);
        append(right, BorderLayout.EAST);

        upButton.setMargin(new Insets(0, 3, 0, 3));
        downButton.setMargin(new Insets(0, 3, 0, 3));
        upButton.setFocusable(false);
        downButton.setFocusable(false);
```

```
        setModel(new DefaultBoundedRangeModel(50, 0, 0, 100));
        textField.setRestrict("0123456789");
        textField.addActionListener(__textActed);
        textField.addEventListener(FocusEvent.FOCUS_OUT,
__textFocusOut);
        upButton.addActionListener(__upValue);
        downButton.addActionListener(__downValue);
        updateTextFromModel();
    }

    public function getModel():BoundedRangeModel{
        return model;
    }

    public function setModel(newModel:BoundedRangeModel):void{
        if (model != null){
            model.removeStateListener(__onModelStateChanged);
        }
        model = newModel;
        if (model != null){
            model.addStateListener(__onModelStateChanged);
        }
    }

    public function setEditable(b:Boolean):void{
        if(editable != b){
            editable = b;
            textField.setEditable(b);
            //使边框得到重绘，以适应新的可编辑属性画面呈现
            repaint();
        }
    }

    public function isEditable():Boolean{
        return editable;
    }

    override public function setEnabled(b:Boolean):void{
        super.setEnabled(b);
        upButton.setEnabled(b);
        downButton.setEnabled(b);
    }
```

```
    private                                                    function
__onModelStateChanged(event:InteractiveEvent):void{
        updateTextFromModel();
        dispatchEvent(new
InteractiveEvent(InteractiveEvent.STATE_CHANGED,
event.isProgrammatic()));
    }

    private function __textActed(e:Event):void{
        updateModelFromText();
    }
    private function __textFocusOut(e:Event):void{
        updateModelFromText();
    }

    private function __upValue(e:Event):void{
        getModel().setValue(getModel().getValue()+1);
    }

    private function __downValue(e:Event):void{
        getModel().setValue(getModel().getValue()-1);
    }

    protected function updateTextFromModel():void{
        textField.setText(getModel().getValue()+"");
    }

    protected function updateModelFromText():void{
        var value:int = parseInt(textField.getText());
        getModel().setValue(value);
    }

    public function addStateListener(listener:Function,
        priority:int=0, useWeakReference:Boolean=false):void{
        addEventListener(InteractiveEvent.STATE_CHANGED,     listener,
false, priority);
    }

    public function removeStateListener(listener:Function):void{
        removeEventListener(InteractiveEvent.STATE_CHANGED, listener);
    }
```

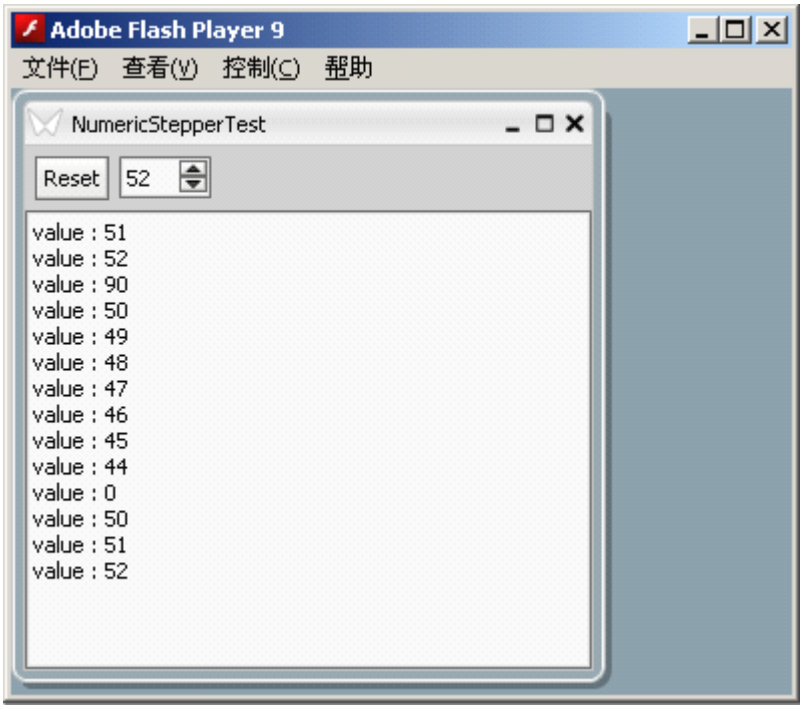
这里的重点，在于给监听模型的 STATE_CHANGED 事件，在模型值改变后，调用 updateTextFromModel 更新文本框的显示，以及监听文本框的输入事件、上下箭头按钮的事件，在事件发生时，设置新的值给模型。这两方面，达到了从显示到模型，从模型到显示的

交互。为了验证这个组件的功能，我们改进测试程序，加入一个按钮设置 NumericStepper 的模型值到默认值 50，然后监听 NumericStepper 模型值变化事件，并输出到一个文本框中，测试程序代码如下：

```
package{

import flash.display.Sprite;
import flash.events.Event;

import org.aswing.AsWingManager;
import org.aswing.BorderLayout;
import org.aswing.JButton;
import org.aswing.JFrame;
import org.aswing.JPanel;
import org.aswing.JScrollPane;
import org.aswing.JTextArea;

public class NumericStepperTest extends Sprite{

    private var infoText:JTextArea;
    private var ns:NumericStepper;

    public function NumericStepperTest(){
        super();
        AsWingManager.initAsStandard(this);
        var frame:JFrame = new JFrame(null, "NumericStepperTest");
        var topPane:JPanel = new JPanel();
        ns = new NumericStepper();
        var button:JButton = new JButton("Reset");
        topPane.appendAll(button, ns);

        infoText = new JTextArea();
        frame.getContentPane().append(topPane, BorderLayout.NORTH);
        frame.getContentPane().append(
            new JScrollPane(infoText), BorderLayout.CENTER);
        frame.setSizeWH(300, 300);
        frame.show();

        ns.addStateListener(__nsStateChanged);
        button.addActionListener(__resetValue);
    }

    private function __resetValue(e:Event):void{
        ns.getModel().setValue(50);
    }
```

```
    private function __nsStateChanged(e:Event):void{
        infoText.appendByReplace("value          :          "          +
ns.getModel().getValue()+"\n");
        infoText.scrollToBottomLeft();
    }
}
}
```

通过随意的几个操作，会得到类似如下的结果：



（图 61）

由此，我们的数字步进器的基本功能算是完成了。剩下的工作，无非是增加一些方法便于获取模型的值，或者再细化文本输入部分，比如支持运行时改变输入框的长度（提供 setColumn 方法），再或者支持像 JAdjuster 那样的 ValueParser 和 ValueTranslator，使之能够显示非数字形式的值。具体实现，读者可以参考 JAdjuster 的源代码自行实验，这里将不多述。

值得一提的是，这个组件完全靠组合搭配现有组件，再加上一个数据模型，就实现出来了，视乎是太简单了？对，大多数自定义组件都只需要这样的步骤就能实现出来，这要归功于 AsWing 基本组件和布局管理器的灵活性。当然，有时候为了实现特殊的表现方式，你还需要编写各种 Icon，Border，GroundDecorator，甚至实现独有的 LayoutManager。归根结底，大部分自定义组件只需要通过**组合搭配现有组件，实现自己的装饰器，或者再加上一个布局管理器，然后再绑上数据模型**，即可完成。

然而，目前这个自定义组件并不完全拥有可替换的 LookAndFeel 能力，这一点将在后面关于如何自定义外观的章节进行补充介绍。

2.9 实现拖拽（DragAndDrop）

现代的应用程序，特别是 RIA 或者游戏，都常常需要拖拽（DragAndDrop）功能给用户带来更好的界面操作体验，比如把一个商品拖入购物车，或者把一个装备从仓库拖入背包，这些都是常见的拖拽功能。AsWing 提供了一套拖拽解决方案，使用基类 Component 的拖拽相关的方法和事件结合 DragManager 工具，可以很方便的实现拖拽功能。

这一节我们以一个简单的拖拽案例来讲解拖拽的实现。

假设我们需要做一个购物界面，商品分为**红色**和**绿色**两种，界面上方是**货架**和**仓库**面板，下方是**绿购物车**和**红购物车**面板，货架和仓库可以存放任何颜色的商品，商品可以随意的放入货架，也可以拖出货架，但仓库只能拖出不能拖入，也就是说，货架可以随意存放商品，但仓库却只能拖商品出来，不能把外面的商品放入仓库。绿购物车只能存放绿色的商品，红购物车只能存放红色的商品，购物车里的商品可以再次被拖回货架（当然，是不能拖回仓库的）。

首先，我们创建商品类，以简单的带色彩的标签来表示商品：

ColorLabel.as

```
package{

import org.aswing.*;
import org.aswing.dnd.*;
import org.aswing.event.DragAndDropEvent;

public class ColorLabel extends JLabel{

    public static const RED:uint = 0xFF0000;
    public static const GREEN:uint = 0x00FF00;
    private var color:uint;

    public function ColorLabel(color:uint, text:String){
        super(text);
        this.color = color;
        setForeground(new ASColor(color));
        setDragEnabled(true);
    }

    public function getColor():uint{
        return color;
    }
}
}
```

它含有一个文本表现，并具有颜色属性以便于后面的拖放判断。然后，我们创建商品面

板类，它是这个程序的重点：

ColorPanel.as

```
package{

import org.aswing.*;
import org.aswing.border.TitledBorder;
import org.aswing.event.DragAndDropEvent;
import org.aswing.util.HashSet;

public class ColorPanel extends JPanel{

    private var dropSet:HashSet;
    private var originalBG:ASColor;

    public function ColorPanel(borderColor:ASColor, title:String,
dropColors:Array){
        super();
        dropSet = new HashSet();
        dropSet.addAll(dropColors);
        var border:TitledBorder = new TitledBorder(null, title);
        border.setColor(borderColor);
        setBorder(border);
        //设置具有拖放探测能力
        setDropTrigger(true);
        setDragAcceptableInitiatorAppraiser(__dropAppraiser);
        addEventListener(DragAndDropEvent.DRAG_ENTER, __dragEnter);
        addEventListener(DragAndDropEvent.DRAG_EXIT, __dragExit);
        addEventListener(DragAndDropEvent.DRAG_DROP, __dragDrop);
        setOpaque(true);
        originalBG = getBackground();
    }

    //是否接受这个被拖入的组件
    private function __dropAppraiser(initiator:ColorLabel):Boolean{
        return dropSet.contains(initiator.getColor());
    }

    //被接受的组件被拖入时，改变Panel背景色
    private function __dragEnter(e:DragAndDropEvent):void{
        var initiator:ColorLabel = e.getDragInitiator() as ColorLabel;
        if(__dropAppraiser(initiator)){
            setBackground(new ASColor(0x0000FF, 0.2));
        }
    }
```

```
    //组件被拖出时，恢复原来的背景色
    private function __dragExit(e:DragAndDropEvent):void{
        setBackground(originalBG);
    }

    //组件被释放时，如果是被接受的，那么把它加入此面板
    private function __dragDrop(e:DragAndDropEvent):void{
        setBackground(originalBG);
        var initiator:ColorLabel = e.getDragInitiator() as ColorLabel;
        if(__dropAppraiser(initiator)){
            append(initiator);
        }
    }
}
}
```

通过程序中的注释我们可以了解到它的工作方式，这里不加赘述。需要解释的是__dragDrop 事件处理方法，此事件在拖拽的组件释放的时候触发，不管面板是否接受此被拖放的组件，这个事件都会触发，因此在处理的时候我们需要再次调用__dropAppraiser 方法判断是否允许放入这个商品。在这个类中，__dropAppraiser 方法起到了非常关键的作用，首先通过 setDragAcceptableInitiatorAppraiser 设置__dropAppraiser 为是否接受释放的判断方法，这是为了拖放管理器在组件拖放过程中调整拖放图形而用，如果拖拽到可接受（也就是__dropAppraiser 返回 true）的面板上时，拖放图形会呈现一个接受释放的图案，反之，则会呈现出一个拒绝图案。对于此判断，也可以通过 *Component.addDragAcceptableInitiator(initator:Component)*添加可接受的组件来达到目的，只不过__dropAppraiser 的方式更为灵活。

然后，我们就可以创建主类来组织这些内容了，主类如下：

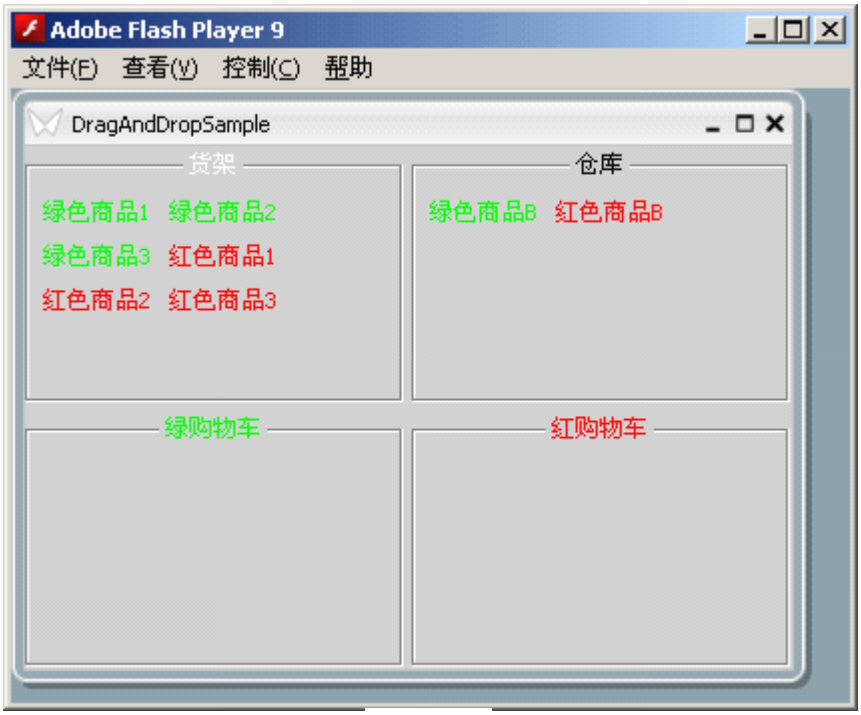
DragAndDropSample.as

```
package{

import flash.display.Sprite;

import org.aswing.*;
import org.aswing.border.TitledBorder;

public class DragAndDropSample extends Sprite{
    private var redArea:JPanel;
    private var greenArea:JPanel;
    private var whiteArea:JPanel;
    private var blackArea:JPanel;

    public function DragAndDropSample(){
        super();
```

```
        AsWingManager.initAsStandard(this);
        var frame:JFrame = new JFrame(this, "DragAndDropSample");
        redArea = new ColorPanel(ASColor.GREEN,
            "绿购物车", [ColorLabel.GREEN]);
        greenArea = new ColorPanel(ASColor.RED,
            "红购物车", [ColorLabel.RED]);
        whiteArea = new ColorPanel(ASColor.WHITE,
            "货架", [ColorLabel.GREEN, ColorLabel.RED]);
        blackArea = new JPanel();
        blackArea.setBorder(new TitledBorder(null, "仓库"));

        var pane:JPanel = new JPanel(new GridLayout(2, 2, 4, 4));
        pane.appendAll(whiteArea, blackArea, redArea, greenArea);

        whiteArea.append(new ColorLabel(ColorLabel.GREEN, "绿色商品1"));
        whiteArea.append(new ColorLabel(ColorLabel.GREEN, "绿色商品2"));
        whiteArea.append(new ColorLabel(ColorLabel.GREEN, "绿色商品3"));
        whiteArea.append(new ColorLabel(ColorLabel.RED, "红色商品1"));
        whiteArea.append(new ColorLabel(ColorLabel.RED, "红色商品2"));
        whiteArea.append(new ColorLabel(ColorLabel.RED, "红色商品3"));
        blackArea.append(new ColorLabel(ColorLabel.GREEN, "绿色商品B"));
        blackArea.append(new ColorLabel(ColorLabel.RED, "红色商品B"));

        frame.setContentPane(pane);
        frame.setSizeWH(400, 300);
        frame.show();
    }
}
}
```

我们为货架添加了 3 个绿色商品，3 个红色商品，给仓库添加了绿红各一个商品，由于仓库不接受拖入商品，因此我们把它创建为简单的 JPanel 即可。

编译并运行主类程序，将看到如下界面：



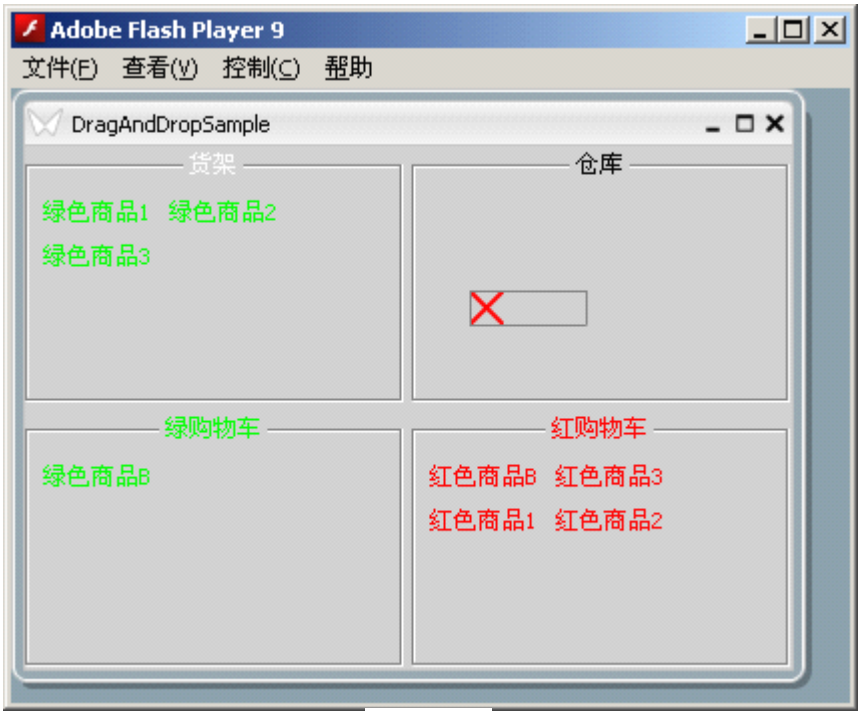
（图 62）

界面没错，但是在拖拽商品的时候，没有任何反应。

这是因为，即使组件被设置 setDragEnabled(true)，拖拽它时也不会自动产生界面反应，它只会触发拖动识别事件，我们监听这个事件并调用 DragManager 的开始拖动方法，启动拖拽行为。给 ColorLabel 监听 DragAndDropEvent.DRAG_RECOGNIZED 事件，并编写事件处理函数如下：

```
private function __dragStart(e:DragAndDropEvent):void{
    //开始拖动，这里的SourceData实际上不会得到使用，
    //不过为了编译通过，我们依然创建了它
    DragManager.startDrag(this,
        new SourceData("color", color));
}
```

再次编译并运行程序，拖动商品，即可得到想要的效果了，下图是拖放了若干商品后，再尝试把绿色商品 3 拖入仓库时的界面表现：



（图 63）

可以发现，拖动图标表现为一个红叉，示意对方拒绝放入。同时我们也发现，这个AsWing默认的拖动图示未能表现出正在拖放的是何种商品。在被接受时，它表现为一个灰色矩形，被拒绝时，表现为矩形+红叉，如果图示也能表现出商品的样子那就更好了。好在 AsWing 支持用户自己实现拖动图示，只要相应地增加 DraggingImage 接口，并在 DragManager 启动拖拽时，传入一个 DraggingImage 的实例即可。我们尝试编写一个能呈现商品原貌的拖动图示类，代码如下：

ColorDragImage.as

```
package{

import flash.display.*;

import org.aswing.*;
import org.aswing.dnd.DraggingImage;
import org.aswing.graphics.*;

public class ColorDragImage implements DraggingImage{

    private var image:Sprite;
    private var source:Bitmap;
    private var width:int;
    private var height:int;

    public function ColorDragImage(dragInitiator:Component){
        width = dragInitiator.width;
        height = dragInitiator.height;
```

```
        source = new Bitmap(new BitmapData(width, height, true, 0x0));
        source.bitmapData.draw(dragInitiator);
        source.alpha = 0.5;
        image = new Sprite();
        image.addChild(source);
    }

    public function getDisplay():DisplayObject{
        return image;
    }

    public function switchToRejectImage():void{
        image.graphics.clear();
        var r:Number = Math.min(width, height) - 2;
        var x:Number = 0;
        var y:Number = 0;
        var g:Graphics2D = new Graphics2D(image.graphics);
        g.drawLine(new Pen(ASColor.RED, 2), x+1, y+1, x+1+r, y+1+r);
        g.drawLine(new Pen(ASColor.RED, 2), x+1+r, y+1, x+1, y+1+r);
    }

    public function switchToAcceptImage():void{
        image.graphics.clear();
    }

}
}
```

它把商品绘制到一个 Bitmap 上，并把这个 Bitmap 加入到拖动图示的元件中，从而重现了商品原貌，修改 ColorLabel 的拖动识别事件处理函数如下：

```
    private function __dragStart(e:DragAndDropEvent):void{
        //开始拖动，这里的SourceData实际上不会得到使用，
        //不过为了编译通过，我们依然创建了它
        DragManager.startDrag(this,
            new SourceData("color", color),
            new ColorDragImage(e.getDragInitiator()));
    }
```

再次编译并运行程序，在尝试把一个绿色商品拖入红色购物车时，将会看到如下界面：

（图 64）

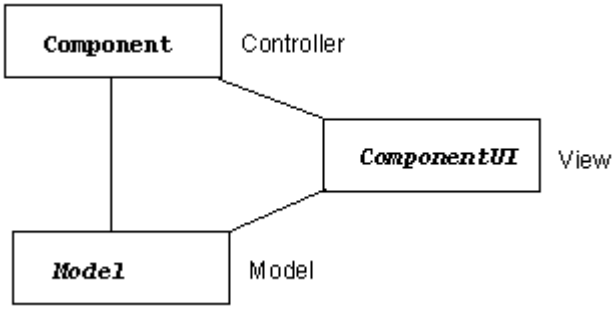
可见，从此用户能明显感受到何种商品被拒绝了。

本例只从最为常用的方式讲述了组件拖拽的实现，一般的拖拽功能采用本例介绍的方法即可满足需求。而实际上，AsWing 对拖放还有更多的支持，组件拖放事件齐备，并且 DragManager 还提供全局的拖放监听，提供全局的拖拽图示控制，具体请参见相关 api 文档。

2.10 自定义观感(LookAndFeel)

在前面的章节中，我们介绍过很多方法来装饰组件，包括设置前景背景色（ASColor），设置字体（ASFont），设置边框（Border），使用图标（Icon），使用前景背景装饰器等（GroundDecorator）。但这些都仅仅是装饰而已，有时候，可能你想要完全改变一个组件的外观，甚至改变交互感，AsWing 框架允许你这样做，并且提供了一套机制让你定制这些内容。这一节，我们将介绍如何改变一个组件乃至一整套组件的观感。

几乎前面所有章节，我们都只是介绍组件的使用，而在定制观感之前，我们不得不先了解一下 AsWing 标准组件的内在结构，如图 65：



（图 65）

大多数标准组件，都具有图中的结构，组件类本身充当控制器（Controller，图中的 Component），它使用一个数据模型（Model，图中的 Model）存储数据，然后用视图（View，图中的 ComponentUI）呈现其外观。AsWing 有一个类存储着所有标准组件的视图集以及所有外观属性，这个类就是 LookAndFeel，UIManager 类管理着 LookAndFeel，当一个组件创建时，就会通过 UIManager 从当前的 LookAndFeel 处获取对应于自己的视图对象。默认的组件外观管理类是 BasicLookAndFeel。

注意：这里讲的标准组件，是指采用了 ComponentUI 来呈现外观的类，大多数 J 字母开头的组件都是标准组件（JRootPane，JPopup，JWindow，JAttachPane，JLoadPane 等由于其自身不需要外观，所以并未使用 ComponentUI，另外 org.aswing.ext 包内的组件也未使用 ComponentUI）。

外观属性配置

要了解 LookAndFeel，必须观察它的源代码，本质上它只是一个静态的属性集合，打开 BasicLookAndFeel 类源代码，你可以看到众多的 Key-Value 键值配置，比如：

```
"window", 0xCCCCCC,
"windowBorder", 0x000000,
"windowText", 0x000000,
"menu", 0xEEEEEE,
"menuText", 0x000000,
```

它配置了窗口（指 JFrame，JPanel 等窗口面板性质的组件）的默认背景色，边框色，

前景文字色，菜单背景色，菜单文字色。这几个属性是为后面具体组件的属性做基础的，比如：

```
            "Panel.background", table.get("window"),
            "Panel.foreground", table.get("windowText"),
            "Panel.opaque", false,
            "Panel.focusable", false,
```

它代表 JPanel 的背景色会使用前面配置好的"window"键的值（即窗口背景色），"Panel.opaque"直接配置了 false，则代表默认情况下 JPanel 是透明的，等等。

假设我们要制作一个自己的 LookAndFeel，采用不同的属性配置，简单的继承自 BasicLookAndFeel，然后覆盖一些属性键的值即可。我们来创建一个简单的 LookAndFeel 只改变普通按钮的前景色（文字颜色）：

OurLookAndFeel1.as

```
package{
import org.aswing.UIDefaults;
import org.aswing.plaf.ASColorUIResource;
import org.aswing.plaf.basic.BasicLookAndFeel;
public class OurLookAndFeel1 extends BasicLookAndFeel{
    public function OurLookAndFeel1(){
        super();
    }
    override protected function initComponentDefaults(
        table:UIDefaults):void{
        super.initComponentDefaults(table);
        var comDefaults:Array = [
            "Button.foreground", new ASColorUIResource(0xFF0000, 1)
        ];
        table.putDefaults(comDefaults);
    }
}
}
```

现在我们覆盖了 initComponentDefaults 方法（这里注意一定要记得调用 super.initComponentDefaults 以保持父类的属性都会被设置到），然后设置了 Button 的前景色为 *new ASColorUIResource(0xFF0000, 1)*。对于颜色，我们没有使用 ASColor 而是选择了 ASColorUIResource，它可以标示出这个属性是属于外观资源的属性。在 LookAndFeel 属性里，我们都尽量使用实现了 UIResource 接口的值类型，以便于组件判断一个属性是否是 LookAndFeel 预设的属性，亦或是应用程序开发者指定的属性。这是因为，当运行中切换 LookAndFeel 时，AsWing 需要知道哪些属性值是需要随着 LookAndFeel 改变，哪些属性需要保持的。比如，旧的 LookAndFeel 让已有的按钮文字颜色是黑色，新的 LookAndFeel 配置给按钮的颜色是红色，但是一部分按钮开发者单独设置了要绿色（当然是采用 ASColor 类的实例来设置的而不是 ASColorUIResource），那么未被开发者单独设置过的按钮，文字颜色会变成红色，单独设置成绿色的则保持不变。

我们编写一个测试程序，使用上面创建的 OurLookAndFeel1，代码如下：

OurLookAndFeel1.as

```
package{
import flash.display.Sprite;
import org.aswing.*;

public class LAFSample1 extends Sprite{
    public function LAFSample1(){
        super();
        AsWingManager.initAsStandard(this);
        //设置LookAndFeel为我们新创建的
        UIManager.setLookAndFeel(new OurLookAndFeel1());
        var frame:JFrame = new JFrame(null, "LAFSample1");
        var button1:JButton = new JButton("Button1");
        var button2:JButton = new JButton("Button2");
        var button3:JToggleButton = new JToggleButton("Button3");
        var pane:JPanel = new JPanel();
        pane.appendAll(button1, button2, button3);
        frame.setContentPane(pane);
        frame.setSizeWH(300, 100);
        frame.show();
    }
}
}
```

编译并运行，将看到如下界面：

（图 66）

可见，普通按钮 JButton 文字变成了红色，而 JToggleButton 并没有受影响，说明我们对“Button.foreground”属性的设置生效了，如果你还想改变 JToggleButton 的颜色，你可以在 OurLookAndFeel1 中也对它的属性进行配置。这里需要注意一点，你需要在创建组件之前就调用 UIManager 来设置 LookAndFeel，否则在设置之前创建的组件时使用的会是旧的 LookAndFeel 配置的属性。当然，如果你不得不在某些组件创建之后才设置新的 LookAndFeel，你可以调用组件的 updateUI 方法来更新它的视图以及所有属性值，如果你需要一次性更新所有已存在组件的视图以及属性值，你可以调用 AsWingUtils.updateAllComponentUI 方法，它会为内存中所有的组件调用 updateUI。

这个例子很简单，我们只修改了一个颜色属性，如果你认真阅读 BasicLookAndFeel 类源代码，你会发现有各种各样的属性都可以设置，包括图标，背景装饰器，边框等等（当然

都是指向了一个实现了 UIResource 的类或实例）。LookAndFeel 具体有哪些外观属性可以配置，这完全取决与这个 LookAndFeel 的整体设计，这跟具体的 ComponentUI 实现相关，后面会讲述，这里读者只需要记住一点：一个 LookAndFeel 需要多少属性，类代码里就会出现多少属性键值对，如果你继承自一个 LookAndFeel，你可以更改属性的值（Value），或者增加属性，但不要删去已有属性。

其实，如果你只是想改变组件的几个外观属性的值，大可不必兴师动众的像上面那样实现一个新的 LookAndFeel，你可以在原本设置新 LookAndFeel 的地方调用如下代码同样可以达到效果：

```
UIManager.getDefaults().put("Button.foreground",
    new ASColorUIResource(0xFF0000, 1));
```

并且不仅仅如此，你还可以给指定的单个组件覆盖属性，不过过程稍微复杂，给上面的测试程序增加一个 JToggleButton，并单独设置外观属性：

```
var button4:JToggleButton = new JToggleButton("Button4");
var ui:ComponentUI = button4.getUI();
ui.putDefault("ToggleButton.foreground",
    new ASColorUIResource(0x00FF00, 1));
ui.putDefault("ToggleButton.highlight",
    new ASColorUIResource(0x0000FF, 1));
button4.setUI(ui);
```

运行后会得到如下效果：

（图 67）

读者可能要问，前景色我直接调用组件的 setForeground 方法就可以设置了，哪用这么麻烦的设置外观属性？完全正确，通用的前景色，背景色，字体等属性都可以通过 Component 提供的方法直接进行设置，但是还有很多 LookAndFeel 特有的属性，必须通过外观属性来进行配置，我们这里就设置了"ToggleButton.highlight"属性来改变按钮边框的亮色。由图可见，Button4 的文字颜色，边线颜色都变了，而同样是 JToggleButton 的 Button3，却没有受到影响。在这个案例中，单独修改一个组件的外观属性，需要给这个组件的 ComponentUI 注入属性值，然后再把这个 ComponentUI 实例设置给组件，此时组件才会得到更新。

实现自己的视图类

到现在，虽然我们通过诸多方式玩转了组件外观，但都还只是改改属性这样的小儿科，下面我们将要深入到内部，创建自己的组建视图（ComponentUI），直接控制组件的绘制，并增加自己的外观属性。假设我们希望我的新的组件外观中， JPanel 的背景色含有一层淡黄

色半透明的过渡色块，由此，我们可以继承 BasicPanelUI 编写我们的 OurPanelUI：

OurPanelUI.as

```
package{

import flash.geom.Matrix;
import org.aswing.*;
import org.aswing.geom.*;
import org.aswing.graphics.*;
import org.aswing.plaf.basic.BasicPanelUI;

public class OurPanelUI extends BasicPanelUI{

    private var gradientColor:ASColor;

    public function OurPanelUI(){
        super();
    }

    override public function installUI(c:Component):void {
        super.installUI(c);
        gradientColor = getColor("Panel.gradientColor");
    }

    override public function uninstallUI(c:Component):void {
        super.uninstallUI(c);
        //这里其实并不需要，因为卸载后，此变量也不会被使用了
        //只是为了规范演示，在做卸载时，我们也做一些工作
        gradientColor = null;
    }

    override protected function paintBackGround(
        c:Component, g:Graphics2D, b:IntRectangle):void{
        if(c.isOpaque()){
            var x:Number = b.x;
            var y:Number = b.y;
            var w:Number = b.width;
            var h:Number = b.height;
            g.fillRectangle(new SolidBrush(c.getBackground()),
                x, y, w, h);

            var rgb:uint = gradientColor.getRGB();
            var alpha:Number = gradientColor.getAlpha();
            var colors:Array = [rgb, rgb];
            var alphas:Array = [alpha, 0];
```

```
            var ratios:Array = [0, 140];
            var matrix:Matrix = new Matrix();
            matrix.createGradientBox(w, h, Math.PI/2, x, y);
           var brush:GradientBrush = new GradientBrush(
            GradientBrush.LINEAR, colors, alphas, ratios, matrix);
           g.fillRectangle(brush, x, y, w, h);
        }
    }
}
}
```

这里我们覆盖了 installUI 方法在其中装载相应的 gradientColor 属性，并覆盖 uninstallUI 方法在卸载时清除资源，覆盖了父类的 paintBackGround 改变背景色绘制策略。然后，再编写 OurLookAndFeel2 并在其中为 OurPanelUI 配置所需的属性：

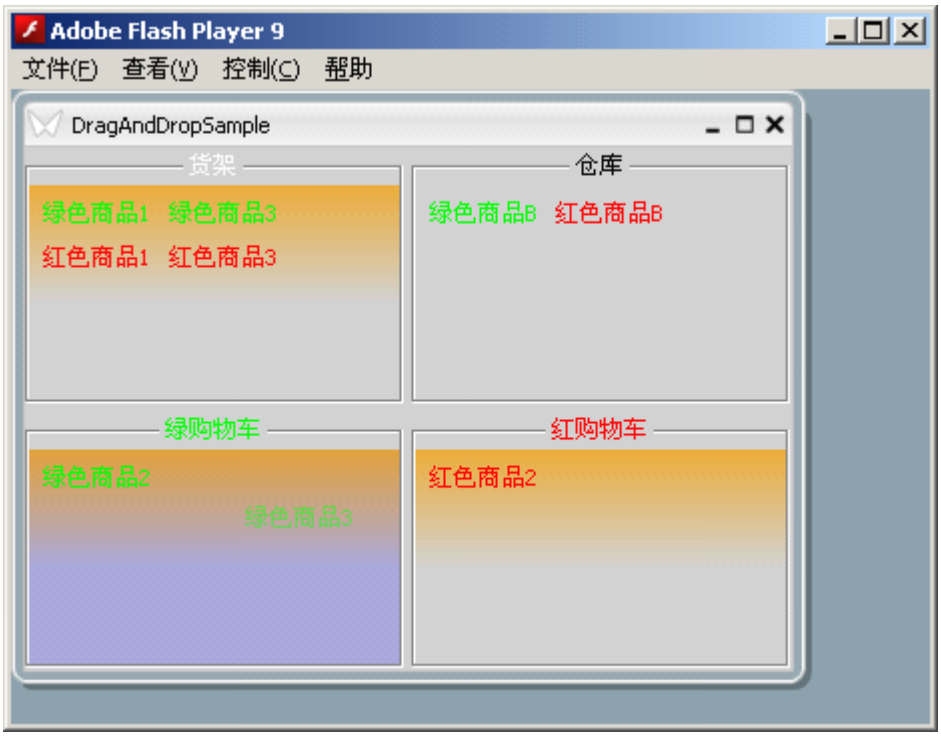
OurPanelUI.as

```
package{

import org.aswing.UIDefaults;
import org.aswing.plaf.ASColorUIResource;
import org.aswing.plaf.basic.BasicLookAndFeel;

public class OurLookAndFeel2 extends BasicLookAndFeel{

    public function OurLookAndFeel2(){
        super();
    }

    override protected function
initClassDefaults(table:UIDefaults):void{
        super.initClassDefaults(table);
        var uiDefaults:Array = [
              "PanelUI", OurPanelUI
          ];
        table.putDefaults(uiDefaults);
    }

    override protected function initComponentDefaults(
        table:UIDefaults):void{
        super.initComponentDefaults(table);
        var comDefaults:Array = [
            "Panel.gradientColor",    new    ASColorUIResource(0xEE9900,
0.75)
        ];
        table.putDefaults(comDefaults);
    }
```

```
}
}
```

注意，淡黄色的"Panel.gradientColor"属性是需要在这个 LookAndFeel 中配置的，当然你也可以写死在 OurPanelUI 类中，如果你愿意失去其灵活性的话。

这里我们让上一节的拖放例子采用这个 LookAndFeel，拖动中，效果如下：



（图 68）

注意，绿购物车的面板颜色与其他有些不同是因为正有绿色商品拖入，此时它被设置背景色为蓝色，加上黄色半透过渡色效果，就呈现为黄紫蓝的过渡效果，仓库面板背景色看起来还是灰色，那是因为仓库面板是透明的（未设置 opaque 为 true），看到的是下部的 JFrame 的灰色底色。

这个例子同样很简单，但是复杂的视图类原理也类似。前面我们尝试过修改单个组件的 ComponentUI 的外观属性值，同理修改单个组件的 ComponentUI 实例也是可以的，如果你没有使用 OurLookAndFeel2，你也可以直接创建一个 OurPanelUI 实例给一个 JPanel（例如：*yourPanel.setUI(new OurPanelUI())*），让它使用 OurPanelUI 来绘制外观，但是需要注意如果没有 OurLookAndFeel2 的支持，OurPanelUI 就获取不到"Panel.gradientColor"属性，你必须通过 OurPanelUI.putDefault 来手动注入。

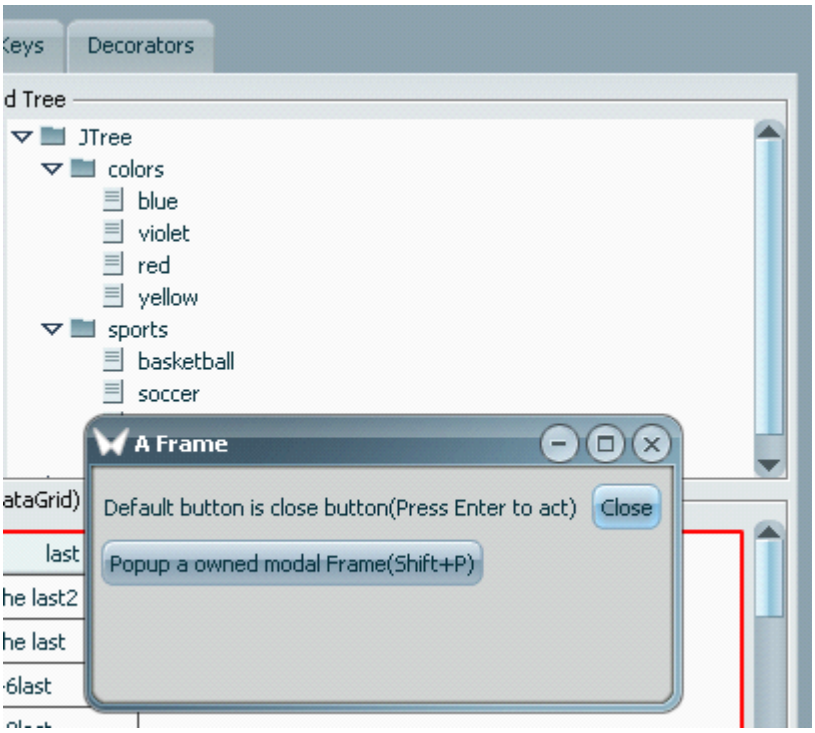
到此，我们介绍了所有改变组件外观的方式，但是LookAndFeel 是指观感，我们只改变了“观”，而并没有接触到“感”。“感”指的是一些互动效果，比如下拉列表拉出列表的动感，你可以通过继承 BasicComboBoxUI 来改变它的下拉速度，对于进度条，你可以改变动画效果，等等，这里就不详细叙述。

皮肤构建器观感

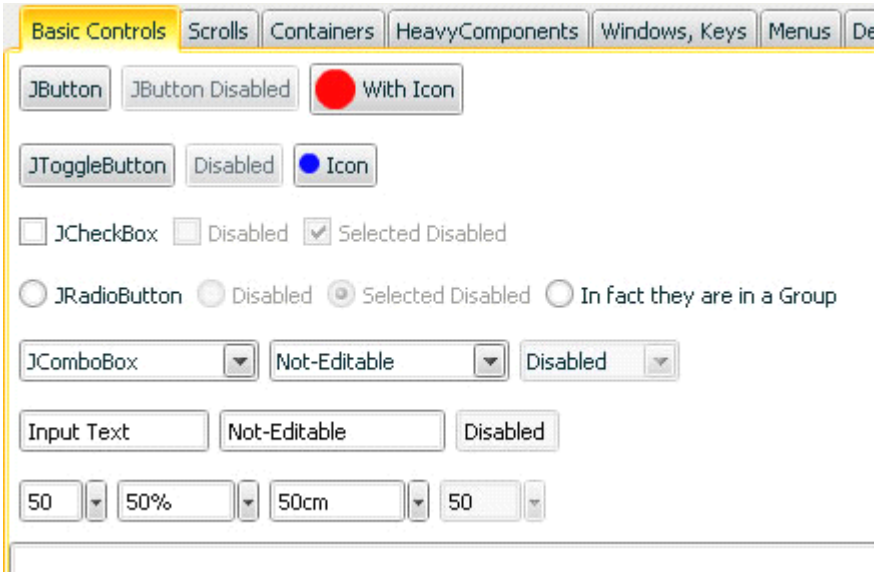
介绍完常规的定制观感之后，这里还不得不提 AsWing 提供的 SkinBuilderLAF,SkinBuilderLAF 本身只是一套自定义LookAndFeel 的实现，但是正如其名，它是以皮肤构建器的目标而设计的。

SkinBuilderLAF 通过组装嵌入的图形资源来表现组件，比如一个按钮，它会通过组装 5 个不同状态的图片资源来表现，普通状态是一个图片，按下状态又会呈现出另一个图片等等，此方式的好处是解放了程序员通过绘图语句来绘制组件外观，直接使用美术设计师通过绘图软件绘制的图形来构建组件外观。

我们先来看看两个用了 SkinBuilderLAF 技术创造的观感的程序界面：



（图 69，默认的 SkinBuilderLAF 实现）



（图 70，AsWing 自带的另一个 SkinBuilderLAF 技术实现的观感 OrangeLookAndFeel）

如果想要让你的程序也使用图 70 的 LookAndFeel，你可以把这个项目包含进你的项目编译路径，然后在程序开头调用 *UIManager.setLookAndFeel(new OrangeLookAndFeel());* 即可。

如果你能找到一个美术设计师为你绘制组件外观，我想你一定忍不住要实现自己的 SkinBuilderLAF，毕竟 AsWing 自带的风格，总是满足不了挑剔的用户、怪僻的设计师，更满足不了你那创意百出的项目经理。于是乎，你开始张罗着配合设计师弄出一个全新的组件外观。

首先，还是得通过查看源代码的方式。打开 SkinBuilderLAF.as，除了与 BasicLookAndFeel 相似的地方外，你还会看到很多这样格式的代码：

```
    [Embed(source="assets/Button_pressedImage.png", scaleGridTop="11",
 scaleGridBottom="12",
        scaleGridLeft="6", scaleGridRight="51")]
    private var Button_pressedImage:Class;

    [Embed(source="assets/Button_rolloverImage.png",
scaleGridTop="11", scaleGridBottom="12",
        scaleGridLeft="6", scaleGridRight="51")]
    private var Button_rolloverImage:Class;
```

这两句是嵌入按钮的按下和鼠标滑过两个状态的图片，Embed 语句中指定了嵌入的图片路径，9 格缩放的缩放点坐标。注意，一定要为需要 9 格缩放的所有图片指定合理的缩放点坐标，否则组件在改变尺寸时，会表现不正确。

相应的，在按钮的外观属配置部分，会看到如下部分代码：

```
            "Button.pressedImage", Button_pressedImage,
            "Button.rolloverImage", Button_rolloverImage,
```

两个属性的值与上面的嵌入图片的类相呼应。由此，即为按钮配置了图片资源（当然按钮并不仅仅只有这两个图片资源，具体请参阅 SkinBuilderLAF 类源代码）。

如果你的设计师为你设计了按钮外观，你需要为不同的状态导出各自的图片，然后覆盖 SkinBuilderLAF 项目中对应的图片文件(assets 目录下的图片文件)，再根据设计师设计的按钮的圆角特性，配置好 9 格缩放点坐标。重新编译 SkinBuilderLAF 项目，即得到你更改后的外观了。

当然，并不是所有图片资源都需要配置 9 格缩放，一些永远不会被缩放的图片，就不需要配置，比如单选框，复选框，窗体的最小化按钮等图片就不需要缩放。它们的嵌入代码类似如下形式：

```
    [Embed(source="assets/RadioButton_defaultImage.png")]
    private var RadioButton_defaultImage:Class;

    [Embed(source="assets/RadioButton_pressedImage.png")]
    private var RadioButton_pressedImage:Class;
```

另外，SkinBuilderLAF 也引入了一些特有的外观属性，比如"Frame.titleBarHeight"，它指定了窗体标题栏的高度，以便于窗体外观能适应窗体背景图片的标题栏部分（窗体被设

计成用一个图片来表现整个窗体，包括标题栏背景和窗体边框）。具体地，在你更改某个部分的图片资源时，详细查看此资源对应的组件外观属性，修改相应参数以适应之。

那么，如果需要为所有组件完全创建一套新的观感，该如何做呢。你可以这样理解，这相当于是只是一一修改各个组件的外观图片，因此简单的方法是，拷贝一整份 SkinBuilderLAF 项目，把 SkinBuilderLAF 类名更改为你想要的名字，然后替换掉所有图片资源文件，修改所有资源对应的 9 格缩放点坐标，适当调整一些外观属性值，即大功告成。这样的行为，相当于把 SkinBuilderLAF 项目作为自建外观的模板来使用。

如果你的设计师能够使用 Flash IDE 来绘制图形，那会变得更方便。你可以用 FlaSkinTemplate_fx 项目来做模板,FlaSkinTemplate_fx 项目与 SkinBuilderLAF 项目原理几乎完全一样，差别只在于 FlaSkinTemplate_fx 嵌入的图片资源不是 png 文件而是 SWF 文件中的元件，嵌入 SWF 中元件的语法形式如下：

```
[Embed(source="assets/Aeon.swf", symbol="Button_pressedImage")]
private var Button_pressedImage:Class;

[Embed(source="assets/Aeon.swf",   symbol="Button_rolloverImage")]

private var Button_rolloverImage:Class;
```

Aeon.swf 是 SWF 文件名，Button_pressedImage 是 Aeon.swf 文件的元件库中绑定名为“Button_pressedImage”的元件。注意这里并没有指定 9 格缩放，这是因为，用 Flash IDE 创建库元件时，可以在 Flash IDE 中就为它设置 9 格缩放，导出的 SWF 文件中的此元件自动就拥有了 9 格缩放能力，在这一点上要比使用 png 图片文件要方便很多。

同样道理，当你要制作自己的组件外观，只需要修改 Aeon.fla，然后发布出新的 Aeon.swf 给 FlaSkinTemplate_fx 项目，重新编译即可。修改单个的 Aeon.fla 总是比修改成百上千的 png 图片要方便很多，如果你的设计师会使用 Flash 来绘制图形，笔者建议你考虑修改 FlaSkinTemplate_fx 项目来制作外观，而不是 SkinBuilderLAF 项目，这会大大提高你们的效率，同时，在体积上 Flash 绘制的矢量图形通常会比 png 位图要小很多，这也为你的项目最终发布的文件减小了尺寸。

注意：Flash CS3 编译器并不支持 Embed 语法，所以以上提到的项目都必须使用 Flex Compiler 来编译。如果你因为某些原因而不得不使用 Flash CS3 来编译你的项目，你可以考虑采用 FlaSkinTemplate_fl 模板，它是 Flash CS3 的兼容方案。(在本书编写到此的时候，Adobe 公司宣布了 Flash CS4，据说此版本已经开始支持 Embed 语法。)

2.11 可视化工具 GuiBuilder

界面的布局工作是枯燥的，大多数成熟的 UI 库都有 IDE 支持 WYSIWYG（所见即所得）的界面创建方式，比如著名的 Visual Basic，Delphi 等 IDE 工具即是以出色的界面布局工具而闻名，Java Swing 也有 NetBeans，Eclipse 的 VE 插件提供布局功能，FlexBuilder 也提供 Design 模式支持直接拖放界面，生成对应的 MXML 代码。

而 AsWing GuiBuilder 则是为 AsWing 量身定做的布局工具，你可以用它创建任何纯 AsWing 界面，给容器添加组件，配置布局，设置组件的属性等，然后还能生成简洁规整的源代码。

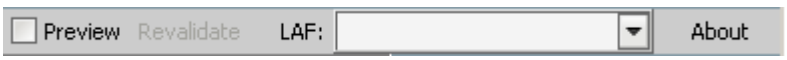
要使用 GuiBuider，首先得安装 Adobe AIR 运行时（你可以从这里下载：http://labs.adobe.com/downloads/air.html）。安装了运行时之后，到 AsWing 下载站点（http://code.google.com/p/aswing/downloads/list）下载最新版的 AsWing GuiBuilder（在本书写到此时，其最新版本是 1.4.2 版本），它是一个 air 安装包，下载后双击即可装上。

首次运行 GuiBuilder，会被要求设置工作目录，选择好工作目录后以后用 GuiBuilder 创建的所有文件将会被保存到工作目录下。GuiBuilder 程序的界面结构如下：

（图 71）

这里先介绍下各区界面的功能点。

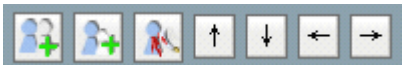
菜单栏：



- New：新建布局文件。
- Open：打开布局文件。

- Save：保存布局文件。
- Close：关闭当前布局文件。
- Generate AS Code：生成当前布局文件的 AS 代码。
- Generate Selection：生成选中组件的 AS 代码。
- View Range：设置是否查看布局的占用范围。
- Preview：设置是否为预览模式，勾选状态为预览模式，否则为编辑模式。
- Revalidate：布局没有自动更新的时候调用，使选中的组件调用 revalidate 方法。
- LAF：切换 GuiBuilder 使用的 LookAndFeel。
- About：查看 GuiBuilder 自述。

操作工具：



从左到右的操作按钮的功能依次为：

- 在选定组件的下面添加一个兄弟组件
- 为选定容器组件的添加一个子组件
- 移除选定组件及其子组件
- 选中组件在兄弟顺序中上移
- 选中组件在兄弟顺序中下移
- 选中组件变为父组件的兄弟
- 选中组件变为其上面兄弟组件的子组件

文件区：

文件区列出了所有打开的布局文件。

组件树：

组件树显示出了当前布局中的所有组件以及其层级关系。

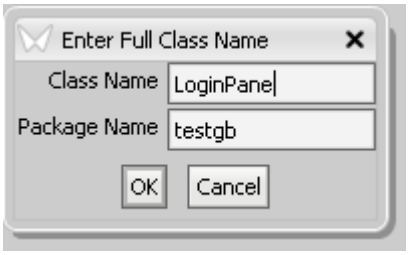
工作/预览区：

布局的内容显示区。在预览模式下，可以操作布局好的组件，在非预览模式下，可以通过点击来选中组件，也可以拖动改变组件的位置和大小（在其容器布局管理器为 EmptyLayout 的情况下改变位置和大小才有效）。

属性区：

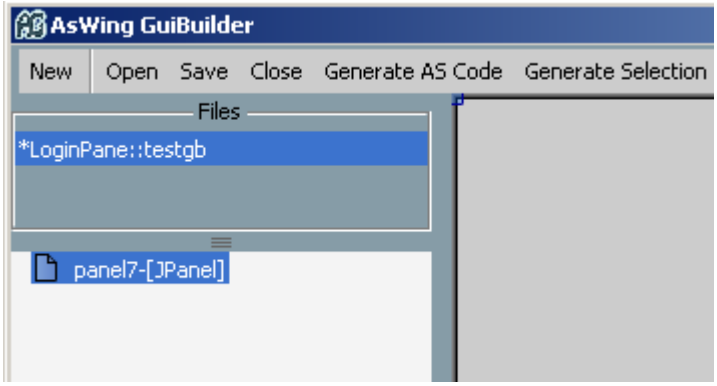
当有组件被选中时，属性区会出现所有此组件可设置属性的设置器，可以通过在此区操作来改变选中组件的各种属性，如字体、背景色、边框、文字内容等等。

现在让我们用 GuiBuilder 来创建一个界面试试吧，假设我们要做一个需要输入用户名和密码的登录框界面。首先，点击菜单栏的 New，新建布局，此时会弹出一个菜单列表，让你选择此布局界面的根容器(由于技术原因，这里并没有提供 AsWing 弹出式容器比如 JPopup 等)，我们选择 JPanel 作为此界面的根容器，然后会弹出对话框询问此界面类的包名和类名，我们填写如下图：



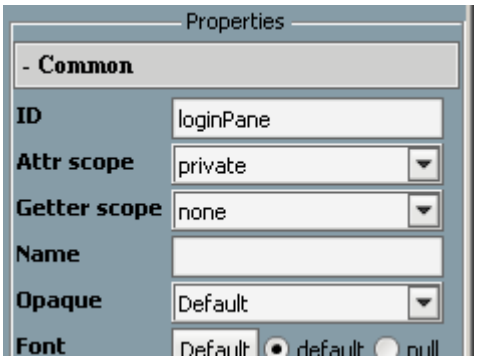
（图 72）

确定后，文件区和组件树区会表现如下：



（图 73）

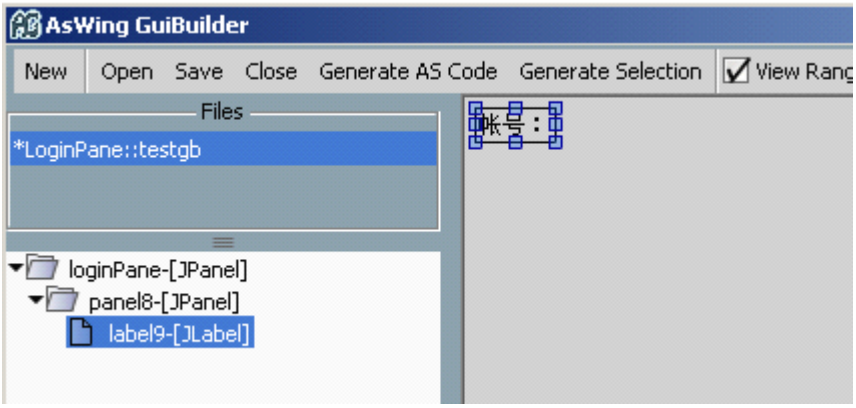
其中*LoginPane::testgb 代表这个布局文件，星号（*）代表布局已做改动，需要保存。组件树中的 panel7-[JPanel]代表根组件 JPanel，panel7 是它的 ID，[JPanel]代表它是一个 JPanel 实例。此时它处于选中状态，右边的属性区会呈现出它的属性设置界面，如下图:



（图 74）

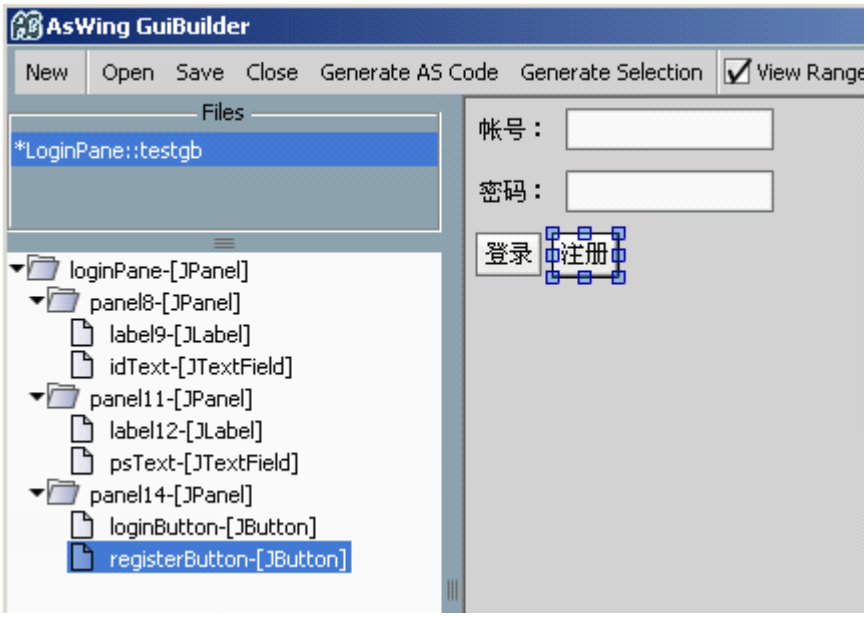
如果我们更改 ID 为 loginPane，组件树中的此节点名称也会自动变为 loginPane-[JPanel]，在属性区，我们还可以设置几乎所有 JPanel 拥有的属性。

下面我们需要给它添加内容了，假设我们用 SoftBoxLayout 来布局它之中的组件，那么点击属性区的 Layout 标签右边的按钮（此时按钮文字为为 Default，代表采用默认布局），弹出布局管理器选择对话框，选择 SoftBoxLayout 并设置 axis 为 Y Axis 并确定。然后点击操作工具的第一个按钮为其添加子组件，此时会弹出组件列表，我们选择 JPanel。现在组件树区域表现为一个 JPanel 含有一个子 JPanel 的状况，子 JPanel 自动被选中，再点击添加子组件按钮添加一个 JLabel，并为 JLabel 设置 Label 属性值为“帐号:”，此时工作区会出现内容，不过是显示为“...”的一个标签，点击菜单栏的 Revalidate 按钮，则会看到完整的标签了，如下图：



（图 75）

以此方式，再为它添加文本输入框，登录按钮等，最后形成如下：



（图 76）

可见登录框成型了，点击 Generate AS Code 菜单栏按钮，得到的代码如下：

LoginPane.as

```
package testgb{

import org.aswing.*;
import org.aswing.border.*;
import org.aswing.geom.*;
import org.aswing.colorchooser.*;
import org.aswing.ext.*;

/**
 * LoginPane
 */
public class LoginPane extends JPanel{

    //members define
```

```
    private var panel8:JPanel;
    private var label9:JLabel;
    private var idText:JTextField;
    private var panel11:JPanel;
    private var label12:JLabel;
    private var psText:JTextField;
    private var panel14:JPanel;
    private var loginButton:JButton;
    private var registerButton:JButton;

    /**
     * LoginPane Constructor
     */
    public function LoginPane(){
        //component creation
        setSize(new IntDimension(400, 400));
        var layout0:SoftBoxLayout = new SoftBoxLayout();
        layout0.setAxis(AsWingConstants.VERTICAL);
        setLayout(layout0);

        panel8 = new JPanel();
        panel8.setSize(new IntDimension(400, 10));

        label9 = new JLabel();
        label9.setLocation(new IntPoint(5, 5));
        label9.setSize(new IntDimension(40, 17));
        label9.setText("帐号：");

        idText = new JTextField();
        idText.setLocation(new IntPoint(50, 5));
        idText.setSize(new IntDimension(88, 21));
        idText.setColumns(12);

        panel11 = new JPanel();
        panel11.setLocation(new IntPoint(0, 31));
        panel11.setSize(new IntDimension(400, 10));

        label12 = new JLabel();
        label12.setLocation(new IntPoint(5, 5));
        label12.setSize(new IntDimension(40, 17));
        label12.setText("密码：");

        psText = new JTextField();
        psText.setLocation(new IntPoint(36, 5));
```

```
        psText.setSize(new IntDimension(104, 21));
        psText.setColumns(12);

        panel14 = new JPanel();
        panel14.setLocation(new IntPoint(0, 62));
        panel14.setSize(new IntDimension(400, 10));

        loginButton = new JButton();
        loginButton.setLocation(new IntPoint(5, 5));
        loginButton.setSize(new IntDimension(31, 22));
        loginButton.setText("登录");

        registerButton = new JButton();
        registerButton.setLocation(new IntPoint(43, 5));
        registerButton.setSize(new IntDimension(33, 22));
        registerButton.setText("注册");

        //component layoution
        append(panel8);
        append(panel11);
        append(panel14);

        panel8.append(label9);
        panel8.append(idText);

        panel11.append(label12);
        panel11.append(psText);

        panel14.append(loginButton);
        panel14.append(registerButton);

    }

    //_________getters_________



    public function getIdText():JTextField{
        return idText;
    }



    public function getPsText():JTextField{
```

```
        return psText;
    }


    public function getLoginButton():JButton{
        return loginButton;
    }

    public function getRegisterButton():JButton{
        return registerButton;
    }


}
}
```

注意组件属性区的 ID，Attr scope，Getter scope 属性的设置和生成的代码之间的关系。Attr scope 指定成员变量的作用域，可选 private，protected，public 或 internal；Getter scope 指定生成的 getXXX 方法的作用域，可选 private，protected，public，internal 或 none（代表不生成 getXXX 方法）；ID 则与直接成为此组件作为成员变量的名和 getXXX 方法的 XXX 名。

我们编写如下测试程序来观察此布局界面在程序中的效果：

```
        AsWingManager.initAsStandard(this);

        var pane:LoginPane = new LoginPane();
        var frame:JFrame = new JFrame(null, "LoginPane");
        frame.setContentPane(pane);
        frame.pack();
        frame.show();
```

测试效果如下：

（图 77）

可见一个简陋的登录框完成得还算不错，如果要获取用户输入的帐号，调用 *pane.getIdText().getText()*即可得到，生成的 LoginPane 类由于是纯界面的代码，如果你保持不直接在代码中修改它，而在外部用另一个控制类操作它，在之后你需要再调整界面时，只需保证若干 getXXX 方法名不变，用调整后生成的新的代码直接覆盖旧的 LoginPane 类，再次编译，即可得到新的界面的程序，做到界面和逻辑的分离。

AsWing GuiBuilder 还有很多细节这里由于篇幅所限，未能详细介绍，不过多试试都能轻松掌握。刚开始使用，比较容易遇到麻烦的地方是 Layout 部分的属性设置，包括组件的坐标，大小，期望大小等属性，需要结合组件布局原理的思想来进行设置，否则可能会难以得到你想要的效果。通常，如果你给容器设置了非 EmptyLayout，你都不用设置位置、大小和期望大小（除非你确定需要），如果你使用了 EmptyLayout，你则可以直接在工作区通过拖动组件范围点来调整组件的位置和大小。GuiBuilder 并不能替代你所有的工作，也难以在未掌握布局原理以及各布局管理器特点的情况下就能方便创建布局，它可以作为你快速布局界面的工具，也可以作为你试验布局的测试器，更能培养你把界面和逻辑分离的习惯。

2.12 AsWing 2.0 预告

将于近期发布，主要改进为：

- 增加组件数字步进器组件 JStepper，增加日期选择组件 JDateChooser。
- 全新的默认外观。
- 新的组件外观自定义方式——通过 StyleTune 来改变组件的样式。
- 更多详细信息请关注 AsWing 网站： http://www.aswing.org（中文网站 http://cn.aswing.org，论坛 http://bbs.aswing.org）

2.13 支持 AsWing 开源项目

如果 AsWing 项目对你有帮助，请支持我们，你可以加入帮助测试程序，在论坛提交 Bug 或建议，分享你的 AsWing 扩展，在你的 Blog 发布你的经验技巧。这些都将帮助 AsWing 更好的发展。

如果 AsWing 帮助你获得了收益，或者其他任何原因你愿意捐款支持 AsWing 的持续发展，我们都将由衷的感谢。每个人的捐赠，都将被列入 aswing.org 站点（如果你愿意的话）。我们目前接受如下方式的捐赠：

通过 **Paypal** 进行捐赠，你可以打开我们的捐赠页面：http://www.aswing.org/?page_id=173 页面的左部有个 Paypal 按钮 

，点击此按钮，即可进行捐赠，视你的经济能力和意愿，金额不限。

通过**支付宝**进行捐赠，登录支付宝 https://www.alipay.com，选择**我要付款**->**即时到帐付款**->**直接给“亲朋好友付钱**或**向陌生人付款**，然后**下一步**，然后输入**收款方支付宝账户**为：*iiley.chen@gmail.com*；**商品或服务名称**为：*AsWing 捐赠*；**商品金额**：视你的经济能力和意愿而定，无下限，高则不要多于 1 万元，因为支付宝有限制；**付款说明**：请填写你希望出现在 aswing.org 捐赠列表的姓名，如果不希望出现，可以不带姓名；**收货地址选择**：*我不需要选择收货地址*。

通过银行转账捐赠，我们暂时不推荐此方法，如果你执意坚持，可以发邮件至 iiley.chen@gmail.com 索取汇款地址或转入银行账号。

所有所得的捐款，将用于以下几种事务的开销：

1. AsWing 网站空间的续费，
2. 可能的 AsWing 教程的英文版翻译费，
3. 捐赠其它开源项目，
4. iiley 写代码时喝的啤酒、抽的香烟钱，
5. 如果还有剩，则再买啤酒和香烟。

最后，谢谢阅读这一节。

完！

2009 年 6 月 8 日
iiley