第 4 章 SWT 图形用户界面

本章要点

1. SWT 程序开发步骤。
2. SWT 常用组件的使用。
3. SWT 的布局。
4. SWT 的事件处理。
5. SWT Designer 简介。

本章难点

1. SWT 常用组件的使用。
2. SWT 的布局。
3. SWT 的事件处理。

4.1SWT/JFace 简介

SWT（Standard Widget Toolkit）即标准小窗口工具箱，是 IBM 公司推出的一种在 Eclipse 中使用的集成开发环境，SWT 提供可移植的 API，并与底层本机 OS GUI 平台紧密集成，它是一个与本地窗口系统集成在一起的小部件集和图形库。SWT 由 JNI（Java Native Interface，Java 本机接口）调用操作系统的内部 API，因此运行速度快，能够获得与操作系统的内部应用程序相同的外观。

JFace 是一个用户界面工具箱，也是一个易用、功能强大的图形包，它简化了常见的图形用户界面的编程任务。SWT 和 JFace 都是 Eclipse 平台上的主要组件。JFace 是在 SWT 的基础上创建的，但 JFace 并不能完全覆盖 SWT 的功能，JFace 和 SWT 的关系如图 4.1 所示。由于 JFace 的功能更强大，因此做图形界面开发时一般优先选用 JFace。

图 4.1 JFace 和 SWT 的关系

4.1.1 SWT 程序开发步骤

在 eclipse 的 plugins 目录下，找到文件 org.eclipse.swt.win32.win32.x86_3.2.1.v3235.jar，文件名中中 3.2.1 是 eclipse 的版本号，v3235 是 SWT 的序列号，不同的 eclipse 版本这两个数字也不同。在 DOS 状态下，用 jar 命令将该文件解压，命令格式如下：

jar xf org.eclipse.swt.win32.win32.x86_3.2.1.v3235.jar

该命令将指定的文件 org.eclipse.swt.win32.win32.x86_3.2.1.v3235.jar 解压到当前目录下。解压后得到四个 DLL 文件：swt-win32-3235.dll，swt-awt-win32-3235.dll，swt-gdip-win32-3235.dll 和 swt-wgl-win32-3235.dll。这四个文件就是 SWT 的原生库文件。原生库文件为 SWT 通过 JNI 访问 windows 本地 API 提供了接口，为使 Java 程序在启动时能够访问这些文件，可以通过以下方法进行设置：

方法一：将这四个 DLL 文件复制到 jre 的 bin 目录下。

方法二：设置环境变量，在 PATH 中加入这几个 dll 文件所在的目录。

方法三：在 eclipse 的 Java 项目中导入原生库文件。操作方法是：

在 eclipse 的包资源管理器中，右单击项目名→导入→常规→文件系统→下一步→浏览→选择 DLL 文件所在目录→确定→勾选 DLL 文件→完成。

导入 SWT 的原生库文件后，还要在 eclipse 的 Java 项目中配置构建路径，添加外部 JAR，将文件 org.eclipse.swt.win32.win32.x86_3.2.1.v3235.jar 加入到项目中，操作方法是：

在 eclipse 的包资源管理器中，右单击项目名→构建路径→配置构建路径→库（L）→添加外部 JAR→在 eclipse 的 plugins 文件夹中找到该 jar 文件→打开→确定。

例 4.1 在 Java 应用程序中使用 SWT 的组件。

操作步骤：

① 新建一个 Java 项目，项目名为：sample4_1。

② 采用方法三在项目中导入原生库文件。

③ 配置构建路径，将 org.eclipse.swt.win32.win32.x86_3.2.1.v3235.jar 加入到项目中。eclipse 包资源管理器可以看到导入的原生库文件和引入的 jar 文件，如图 4.2 所示。

④ 在项目中新建一个类，文件名为 HelloSWT.java。

⑤ 在类文件中写入代码。

图 4.2 包资源管理器

图 4.3 程序运行结果

HelloSWT.java 文件内容如下：

```
package edu.ch4;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Text;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.graphics.*;
 class HelloSWT {


    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        Shell shell=new Shell(display);//shell是程序的主窗体
```

```
        shell.setLayout(null);          //设置shell的布局方式
        Text hello=new Text(shell,SWT.MULTI);  //声明一个可以显示多行信息的文本框
        shell.setText("Java应用程序");  //设置主窗体的标题
        shell.setSize(200,100);         //设置主窗体的大小
        Color color=new Color(Display.getCurrent(),255,255,255);//声明颜色对象
        shell.setBackground(color);    //设置窗体的背景颜色
        hello.setText("Hello, SWT World!\n\n你好，SWT世界！");//设置文本框信息
        hello.pack();     //自动调整文本框的大小
        //shell.pack();  //自动调整主窗体的大小
        shell.open();     //打开主窗体
        while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){  //如果display不忙
            display.sleep();     //休眠
            }
        }
        display.dispose();       //销毁display
    }
    }
```

在包资源管理器中，右单击文件名 HelloSWT.java→运行方式→Java 应用程序，程序运行结果如图 4.3 所示。该窗体具有典型的 Windows 风格。

分析本例的源代码，可以看到，创建一个典型的 SWT 应用程序需要以下步骤：

①创建一个 Display

②创建一个或多个 Shell

③设置 Shell 的布局

④创建 Shell 中的组件

⑤用 open()方法打开 Shell 窗体

⑥写一个事件转发循环

⑦销毁 display

4.1.2 SWT 中的包

SWT 是 Eclipse 图形 API 的基础，本节简单介绍一下 SWT 中常用的包。

1. org.eclipse.swt.widgets

最常用的组件基本都在此包中，如 Button、Text、Label、Combo 等。其中两个最重要的组件是 Shell 和 Composite。Shell 相当于应用程序的主窗体；Composite 是容纳组件的容器，相当于 SWING 中的 Panel 对象。

2. org.eclipse.swt.layout

主要的界面布局方式在此包中。SWT 对组件的布局也采用了 AWT/SWING 中的 Layout 和 Layout Data 结合的方式。

3. org.eclipse.swt.custom

对一些基本图形组件的扩展在此包中，比如其中的 CLabel 就是对标准 Label 组件的扩展，在 CLabel 上可以同时加入文字和图片。在此包中还有一个新的布局方式 StackLayout。

4. org.eclipse.swt.event

SWT 采用了和 AWT/SWING 一样的事件模型，在包中可以找到事件监听类和相应的事

件对象。比如，鼠标事件监听器 MouseListener，MouseMoveListener 等，及对应的事件对象 MouseEvent。

5. org.eclipse.swt.graphics

此包中包含针对图片、光标、字体或绘图 API。比如，可通过 Image 类调用系统中不同类型的图片文件。

6. org.eclipse.swt.ole.win32

对不同平台，SWT 有一些针对性的 API。例如，在 Windows 平台，可以通过此包很容易的调用 OLE 组件，这使得 SWT 程序也可以内嵌 IE 浏览器或 Word、Excel 等程序。

4.2 SWT/JFace 常用组件

SWT/JFace 常用组件有按钮（Button 类）、标签（Label 类）、文本框（Text 类）、下拉框（Combo 类）和列表框（List 类）等。

4.2.1 按钮组件

按钮（Button）组件是 SWT 中最常用的组件，Button 类的构造方法是：

Button(Composite parent,int style)

该方法有两个参数：

第一个参数 parent 是指 Button 创建在哪一个容器上。Composite（面板）是最常用的容器，Shell（窗体）继承自 Composite，此参数也能接受 Shell 和任何继承自 Compsite 的类。第二个参数 style 用来指定 Button 的式样。SWT 组件可以在构造方法中使用式样（style）来声明组件的外观形状和文字的式样。SWT 组件的构造方法和 Button 类相似，参数的含义也相同。

1．Button 组件常用式样

SWT.PUSH：按钮。

SWT.CHECK：多选按钮。

SWT.RADIO：单选按钮。

SWT.ARROW：箭头按钮。

SWT.NONE：默认按钮。

SWT.CENTER：文字居中，与 SWT.NONE 相同。

SWT.LEFT：文字靠左。

SWT.RIGHT：文字靠右。

SWT.BORDER：深陷型按钮。

SWT.FLAT：平面型按钮。

一个 Button 也可以指定多个式样，只要将指定的各个式样用符号“|”连接起来即可。如：

Button bt=new Button(shell,SWT.CHECK|SWT.BORDER|SWT.LEFT);

表示创建的按钮 bt 是一个复选按钮（CHECK），深陷型（BORDER）、文字左对齐（LEFT）。

2．Button 组件的常用方法

setText(String string)：设置组件的标签文字。

setBounds(int x,int y,int width,int height)：设置组件的坐标位置和大小（x 轴坐标，y 轴坐标，组件宽度 width，组件高度 height）。

setEnabled(Boolean enabled)：设置组件是否可用。true：可用（默认值），false：不可用。

setFont(Font font)：设置文字的字体。

setForeground(Color color)：设置前景色。

setBackgrount(Color color)：设置背景色。

setImage(Image image)：设置显示的图片。

setSelection(Boolean selected)：设置是否选中（仅对复选框或单选框有效）。true：选中，false：未选中（默认值）。

setToolTipText(String string)：设置鼠标停留在组件上时出现的提示信息。

以上方法在其他组件中也可使用。

例 4.2 按钮示例。

按照例 4.1 的操作步骤建立项目、设置构建路径和引入原生库。类 Sample4_2.java 源代码如下：

```
package edu.ch4;
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
public class Sample4_2 {
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        Shell shell=new Shell(display);//shell是程序的主窗体
        //shell.setLayout(null);          //设置shell的布局方式
        shell.setText("按钮示例");  //设置主窗体的标题
        Button bt1=new Button(shell,SWT.NULL);  //创建默认按钮
        bt1.setText("SWT.NULL");            //设置按钮上的文字
        bt1.setBounds(10,10,75,30);         //设置按钮显示位置及宽度、高度
        Button bt2=new Button(shell,SWT.PUSH|SWT.BORDER); //创建深陷型按钮
        bt2.setText("SWT.PUSH");
        bt2.setBounds(90,10,75,30);
        Button check1=new Button(shell,SWT.CHECK);//创建复选按钮
        check1.setText("SWT.CHECK");
        check1.setBounds(10,50,75,30);
        Button check2=new Button(shell,SWT.CHECK|SWT.BORDER);//创建深陷型复选按钮
        check2.setText("SWT.CHECK");
        check2.setBounds(90,50,75,30);
        Button radio1=new Button(shell,SWT.RADIO);//创建单选按钮
        radio1.setText("SWT.RADIO");
        radio1.setBounds(10,90,75,30);
        Button radio2=new Button(shell,SWT.RADIO|SWT.BORDER);//创建深陷型单选按钮
        radio2.setText("SWT.RADIO");
        radio2.setBounds(90,90,75,30);
        Button arrowLeft=new Button(shell,SWT.ARROW|SWT.LEFT);//创建箭头按钮（向左）
        arrowLeft.setBounds(10,130,75,20);
        Button arrowRight=new Button(shell,SWT.ARROW|SWT.RIGHT|SWT.BORDER);
        arrowRight.setBounds(90,130,75,20);
        shell.pack();  //自动调整主窗体的大小
```

```
            shell.open();     //打开主窗体
            while(!shell.isDisposed()){   //如果主窗体没有关闭
                if(!display.readAndDispatch()){   //如果display不忙
                display.sleep();     //休眠
                }
            }
            display.dispose();        //销毁display
        }
}
```

运行结果如图 4.4 所示。

图 4.4 按钮

图 4.5 标签

4.2.2 标签组件

标签（Label 类）组件是 SWT 中最简单的组件。Label 类的构造方法和 Button 类相似，参数的含义与相同，格式如下：

Label(Composite parent,int style)

Label 类的常用式样有以下几种：

Label 类常用的式样如下：

SWT.CENTER：文字居中。

SWT.RIGHT：文字靠右。

SWT.LEFT：文字靠左。

SWT.NONE：默认式样。

SWT.WRAP：自动换行。

SWT.BORDER：深陷型。

SWT.SEPARATOR：分栏符，默认为竖线分栏。

SWT.HORIZONTAL：横线分栏符。

例 4.3 标签示例。

```
package edu.ch4;
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.Font;
public class Sample4_3 {
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        Shell shell=new Shell(display);//shell是程序的主窗体
        //shell.setLayout(null);          //设置shell的布局方式
        shell.setText("标签示例");  //设置主窗体的标题
```

```
        Label lb1=new Label(shell,SWT.BORDER|SWT.RIGHT);//深陷型、文字右对齐
        lb1.setBounds(10,10,70,30);
        lb1.setText("标签1");
        lb1.setFont(new Font(display,"黑体",14,SWT.BOLD));//设置文字的字体字号
        lb1.setForeground(Display.getCurrent().getSystemColor(SWT.COLOR_BLUE));
        Label lb2=new Label(shell,SWT.CENTER);//文字居中的标签
        lb2.setBounds(90,10,70,30);
        lb2.setText("标签2");
        lb2.setFont(new Font(display,"宋体",14,SWT.NORMAL));//设置文字的字体字号
        Label lb3=new Label(shell,SWT.SEPARATOR|SWT.BORDER);//竖直分栏符
        lb3.setBounds(10,50,70,30);
        Label lb4=new Label(shell,SWT.SEPARATOR|SWT.HORIZONTAL|SWT.BORDER);
        //水平分栏符
        lb4.setBounds(90,50,70,30);
        shell.pack();  //自动调整主窗体的大小
        shell.open();    //打开主窗体
        while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){  //如果display不忙
            display.sleep();    //休眠
            }
        }
        display.dispose();      //销毁display
    }
}
```

程序运行结果如图4.5所示。

4.2.3 文本框组件

文本框（Text 类）的式样如下：

SWT.NONE：默认式样。

SWT.CENTER：文字居中。

SWT.LEFT：文字靠左。

SWT.RIGHT：文字靠右。

SWT.MULTI：可以输入多行，须回车换行。

SWT.WRAP：可以输入多行，到行尾后自动换行。

SWT.PASSWORD：密码型，输入字符显示成“*”。

SWT.BORDER：深陷型。

SWT.V_SCROLL：带垂直滚动条。

SWT.H_SCROLL：带水平滚动条。

例 4.4 各种文本框示例。

```
package edu.ch4;
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
public class Sample4_4 {
```

```
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        Shell shell=new Shell(display);//shell是程序的主窗体
        shell.setText("文本框示例");
        Text text1=new Text(shell,SWT.NONE|SWT.BORDER);//带边框
        text1.setBounds(10,10,70,30);
        Text text2=new Text(shell,SWT.PASSWORD);
        text2.setBounds(90,10,70,30);
        Text text3=new Text(shell,SWT.MULTI|SWT.V_SCROLL|SWT.H_SCROLL);
        text3.setBounds(10,50,70,70);
        Text text4=new Text(shell,SWT.WRAP|SWT.V_SCROLL);
        text4.setBounds(90,50,70,70);
        shell.pack();
        shell.open();
        while(!shell.isDisposed()){   //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){   //如果display不忙
            display.sleep();      //休眠
            }
        }
        display.dispose();        //销毁display
    }
}
```

运行结果如图 4.6 所示。

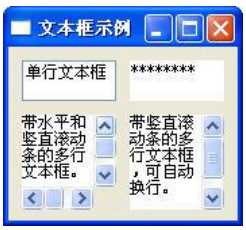

图 4.6 文本框

图 4.7 下拉框

4.2.4 下拉框组件

1．下拉框（Combo 类）的式样

SWT.NONE：默认式样。

SWT.READ_ONLY：只读。

SWT.SIMPLE：无须单击下拉框，列表会一直显示。

2．下拉框（Combo 类）的常用方法

add(String string)：在 Combo 中增加一项。

add(String string,int index)：在 Combo 的第 index 项后插入一项。

deselectAll()：使 Combo 组件中的当前选择项置空。

removeAll()：将 Combo 中的所有选项清空。

setItems(String[] items)：将数组中的各项依次加入到 Combo 中。

select(int index)：将 Combo 的第 index+1 项设置为当前选择项。

例 4.5 下拉框示例。

```
package edu.ch4;
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;
public class Sample4_5 {
    private static Label lb;
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        final Shell shell=new Shell(display);//shell是程序的主窗体
        shell.setText("下拉框示例");
        final Combo combo=new Combo(shell,SWT.NONE);
        combo.setBounds(10,10,100,25);
        lb=new Label(shell,SWT.WRAP);   //创建标签，可自动换行
        lb.setBounds(120,10,100,35);
        Button bt1=new Button(shell,SWT.NONE);
        bt1.setBounds(20,60,100,25);
        bt1.setText("设值");
        bt1.addSelectionListener(new SelectionAdapter(){
            public void widgetSelected(SelectionEvent e){ //按钮的单击事件
                combo.removeAll();        //清空combo
                for(int i=1;i<=3;i++){
                    combo.add("第"+i+"项");  //循环添加选项
                }
                combo.select(0);    //设置默认选项
            }
        });
        Button bt2=new Button(shell,SWT.NONE);
        bt2.setBounds(130,60,100,25);
        bt2.setText("取值");
        bt2.addSelectionListener(new SelectionAdapter(){
            public void widgetSelected(SelectionEvent e){ //按钮的单击事件
                lb.setText("你选择的是："+combo.getText());
            }
        });
        shell.pack();
        shell.open();
        while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){  //如果display不忙
            display.sleep();    //休眠
            }
        }
        display.dispose();      //销毁display
    }
```

}

运行结果如图 4.7 所示。本例中，按钮 bt1 和 bt2 添加了按钮选择监听事件代码。下拉框最初没有值，单击【设置】按钮后将一组数据加入，单击【取值】按钮，标签 lb 显示取值的结果。

4.2.5 列表框组件

列表框（List 类）组件的用法和下拉框（Combo 类）相似。

1．列表框（List 类）的式样

SWT.NONE：默认式样。

SWT.V_SCROLL：带垂直滚动条。

SWT.MULTI：允许复选。

SWT.SINGLE：允许单选。

2．常用方法

列表框（List 类）组件的方法和下拉框（Combo 类）是一样的，但由于 List 可选择多项，而 Combo 只能选择一项，所以 List 没有 getText()方法，List 的取值是用 getSelection()方法，返回一个所有选项组成的 String 数组。

例 4.6 列表框示例。

```
package edu.ch4;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.List;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
public class Sample4_6 {
    private static Label lb;
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        final Shell shell=new Shell(display);//shell是程序的主窗体
        shell.setText("列表框示例");
        final List list=new List(shell,SWT.MULTI|SWT.V_SCROLL|SWT.BORDER);
         //声明一个可复选、带垂直滚动条、有边框的列表框。
        list.setBounds(10,10,100,50);
        lb=new Label(shell,SWT.WRAP);
        lb.setBounds(120,10,90,50);
        Button bt1=new Button(shell,SWT.NONE);
        bt1.setBounds(20,60,100,25);
        bt1.setText("设值");
        bt1.addSelectionListener(new SelectionAdapter(){
            public void widgetSelected(SelectionEvent e){
                list.removeAll();
```

```
                for(int i=1;i<=8;i++){
                    list.add("第"+i+"项");    //将选项循环加入到列表框中
                }
                list.select(0);
            }
        });
        Button bt2=new Button(shell,SWT.NONE);
        bt2.setBounds(130,60,100,25);
        bt2.setText("取值");
        bt2.addSelectionListener(new SelectionAdapter(){
            public void widgetSelected(SelectionEvent e){
                String[] selected=list.getSelection(); //声明字符串数组保存选项
                String outStr=" ";
                for(int j=0;j<selected.length;j++){
                    outStr=outStr+" "+selected[j]; //将数组中的选项加入到输出字符串
                }
                lb.setText("你选择的是："+outStr);
            }
        });
        shell.pack();
        shell.open();
        while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){  //如果display不忙
            display.sleep();    //休眠
            }
        }
        display.dispose();      //销毁display
    }
}
```

本例开始运行时，列表框是空的，单击【设置】按钮，将选项加入到列表框中，按【Ctrl】键加鼠标左键选项，再按【取值】按钮。运行结果如图 4.8 所示。

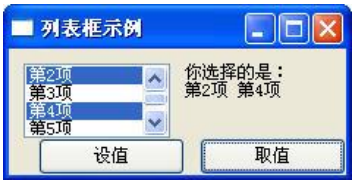

图 4.8 列表框

4.2.6 菜单

菜单（Menu 类，MenuItem 类）是常用的 SWT 组件，Menu 是一个菜单栏，同时也是一个容器，可以容纳菜单项（MenuItem）。

1．Menu 的式样

SWT.BAR：菜单栏，用于主菜单。
SWT.DROP_DOWN：下拉菜单，用于子菜单。
SWT.POP_UP：鼠标右键弹出式菜单。
2．MenuItem 的式样
SWT.CASCADE：有子菜单的菜单项。
SWT.CHECK：选中后前面显示一个小勾。
SWT.PUSH：普通型菜单。
SWT.RADIO：选中后前面显示一个圆点。
SWT.SEPARATOR：分隔符。
3．建立菜单的一般步骤：
①首先建立一个菜单栏，需要使用 SWT.BAR 属性。

```
Menu mainMenu=new Menu(shell,SWT.BAR);
```

②在窗体中指定需要显示的菜单栏。

```
shell.setMenuBar(mainMenu);
```

③创建顶级菜单项，需要使用 SWT.CASCADE 属性。

```
MenuItem fileItem=new MenuItem(mainMenu,SWT.CASCADE);
fileItem.setText("文件&F");
```

④创建与顶级菜单项相关的下拉式菜单。

```
Menu fileMenu=new Menu(shell,SWT.DROP_DOWN);
```

⑤将顶级菜单项与下拉菜单关联。

```
fileItem.setMenu(fileMenu);
```

二级菜单的创建只需重复以上步骤③～⑤。注意：本例创建所有 Menu 对象的第一个参数都是 shell；创建 MenuItem 对象的第一个参数是该 MenuItem 所在的 Menu 对象；如果某 Menu 是某 MenuItem 的子菜单，则还要建立关联：MenuItem.setMenu(Menu)。源代码如下：

```
package edu.ch4;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.widgets.MenuItem;
import org.eclipse.swt.events.*;
import org.eclipse.swt.*;
public class Sample4_7 {
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        final Shell shell=new Shell(display);//shell是程序的主窗体
        shell.setText("菜单示例");
        Menu mainMenu=new Menu(shell,SWT.BAR);
        shell.setMenuBar(mainMenu);
        //Menu mainMenu=new Menu(shell,SWT.POP_UP); //创建弹出式菜单
        //shell.setMenu(mainMenu);    //创建弹出式菜单
            {
            //"文件"项
            MenuItem fileItem=new MenuItem(mainMenu,SWT.CASCADE);
            fileItem.setText("文件&F");
            //"文件"菜单
```

```
            Menu fileMenu=new Menu(shell,SWT.DROP_DOWN);
            fileItem.setMenu(fileMenu);
            {
                //"新建"项
                MenuItem newFileItem=new MenuItem(fileMenu,SWT.CASCADE);
                newFileItem.setText("新建&N");
                //"新建"菜单
                Menu newFileMenu=new Menu(shell,SWT.DROP_DOWN);
                newFileItem.setMenu(newFileMenu);
                {
                    //"新建项目"项
                    MenuItem newProjectItem=new MenuItem(newFileMenu,SWT.PUSH);
                    newProjectItem.setText("项目\tCtrl+Shift+N");
                    //设置快捷键
                    newProjectItem.setAccelerator(SWT.CTRL+SWT.SHIFT+'N');
                    //添加事件监听
                    newProjectItem.addSelectionListener(new SelectionAdapter(){
                        public void widgetSelected(SelectionEvent e){
                        Text text=new Text(shell,SWT.MULTI|SWT.BORDER|SWT.WRAP);
                            text.setBounds(10,10,100,30);
                            text.setText("你选择了“新建项目”");
                        }
                    });
                    new MenuItem(newFileMenu,SWT.SEPARATOR);
                    new MenuItem(newFileMenu,SWT.PUSH).setText("包");
                    new MenuItem(newFileMenu,SWT.PUSH).setText("类");
                }
                MenuItem openFileItem=new MenuItem(fileMenu,SWT.CASCADE);
                openFileItem.setText("打开&O");
                MenuItem exitItem=new MenuItem(fileMenu,SWT.CASCADE);
                exitItem.setText("退出&E");
            }
            MenuItem helpItem=new MenuItem(mainMenu,SWT.CASCADE);
            helpItem.setText("帮助&H");
        }
        shell.pack();
        shell.open();
        while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){  //如果display不忙
            display.sleep();    //休眠
            }
        }
        display.dispose();      //销毁display
    }
```

}

程序运行结果如图 4.9、4.10 所示。当点击【文件】→【新建】→【项目 Ctrl+Shift+N】时，文本框中显示“你选择了‘新建项目’”。

图 4.9 选择菜单

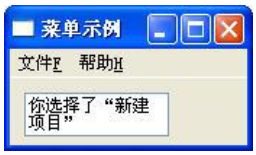

图 4.10 运行结果

创建弹出式菜单只需将①、②两步的代码改为以下两行即可。

```
Menu mainMenu=new Menu(shell,SWT.POP_UP); //创建弹出式菜单
shell.setMenu(mainMenu);    //创建弹出式菜单
```

4.3 容器类

通常，组件构建在容器类中，容器构建在主窗体（shell）中，主窗体也是容器，也就是说，容器不仅可以容纳组件，也可以容纳容器。有了容器，就可以通过它来对组件进行集体操作。例如，容器在界面上移动时，其上的组件也会随着容器移动，容器隐藏，其组件也会被隐藏，容器销毁（dispose），其组件也会被销毁。

4.3.1 面板

面板（Composite 类）是最常用的容器。主窗体（shell）是面板（Composite）的子类。面板的构造方法格式如下：

Composite(Composite parent,int style)

第一个参数表示该容器创建在哪个容器上，第二个参数表示容器的式样。Composite 的式样一般都是用 SWT.NONE，这时 Composite 在界面是不显示出来的，只是发挥着容器的作用。如果要让容器形成凹陷效果，可以用 SWT.BORDER 式样。例如，在主窗体中创建一个容器：

Composite composite=new Composite(shell,SWT.NONE);

Composite 的常用方法：

getLayout()：得到布局管理器。

getLayoutData()：得到布局数据。

getParent()：得到容纳该容器的父容器。

getShell()：得到容纳该容器的 Shell。

layout()：将容器上的组件重新布局，相当于刷新。

例 4.8 面板示例。

```
package edu.ch4;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
```

```
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
public class Sample4_8 {
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        final Shell shell=new Shell(display);//shell是程序的主窗体
        shell.setText("容器示例");
        Composite composite1=new Composite(shell,SWT.NONE);
        composite1.setBounds(10,10,100,50);
        Composite composite2=new Composite(shell,SWT.BORDER);
        composite2.setBounds(120,10,100,50);
        Label lb1=new Label(composite1,SWT.NONE);
        lb1.setText("面板1");
        lb1.pack();
        Label lb2=new Label(composite2,SWT.NONE);
        lb2.setText("面板2");
        lb2.pack();
        shell.pack();
        shell.open();
        while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){  //如果display不忙
            display.sleep();     //休眠
            }
        }
        display.dispose();       //销毁display
    }
}
```

运行结果如图 4.11 所示。

图 4.11 面板

图 4.12 分组框

4.3.2 分组框

分组框（Group 类）是面板（Composite 类）的子类，所以两者用法基本相同。主要区别是 Group 显示有一个方框，且方框线上还可以显示说明文字。

例 4.9 分组框示例。

```
package edu.ch4;
import org.eclipse.swt.SWT;
```

```
import org.eclipse.swt.widgets.*;
public class Sample4_9 {
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        final Shell shell=new Shell(display);//shell是程序的主窗体
        shell.setText("分组框示例");
        Group group1=new Group(shell,SWT.NONE);  //创建分组框
        group1.setText("录入信息");              //设置分组框说明信息
        group1.setBounds(10,20,200,100);
        Label lb1=new Label(group1,SWT.NONE);  //在分组框中加入组件
        lb1.setText("姓名：");
        lb1.setBounds(10,20,70,20);
        Text text1=new Text(group1,SWT.BORDER);
        text1.setBounds(90,20,70,20);
        Label lb2=new Label(group1,SWT.NONE);
        lb2.setText("地址：");
        lb2.setBounds(10,50,70,20);
        Text text2=new Text(group1,SWT.BORDER);
        text2.setBounds(90,50,70,20);
        shell.pack();
        shell.open();
        while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){  //如果display不忙
            display.sleep();    //休眠
            }
        }
        display.dispose();      //销毁display
    }
}
```

运行结果如图 4.12 所示。

4.3.3 选项卡

选项卡包括一个选项卡（TabFolder 类）和一个选项页（TabItem 类），TabFolder 是容器，可以容纳其他容器和组件，但 TabItem 不是容器，可以把它看成是一个选项标签，TabFolder 通过 TabItem 来对其中的组件进行控制。每一个 TabItem 用 setControl()方法来控制一个界面组件。

例 4.10 选项卡示例。

```
package edu.ch4;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.*;
public class Sample4_10 {
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
```

```
final Shell shell=new Shell(display);//shell是程序的主窗体
shell.setText("选项卡示例");
TabFolder tabFolder=new TabFolder(shell,SWT.NONE);//声明一个选项卡容器
tabFolder.setBounds(5,5,180,130);  //设置选项卡的位置和大小
TabItem tabItem1=new TabItem(tabFolder,SWT.NONE);//声明第1个选项页
tabItem1.setText("选项1");   //设置选项页的标题
{
     //创建第1个分组框，建立在tabFolder上
    Group group1=new Group(tabFolder,SWT.NONE);
    group1.setText("录入信息");              //设置分组框说明信息
    tabItem1.setControl(group1);     //让tabItem1控制group1
    Label lb1=new Label(group1,SWT.NONE);  //注意Label建立在group1上
    lb1.setText("姓名：");
    lb1.setBounds(10,20,70,20);
    Text text1=new Text(group1,SWT.BORDER);
    text1.setBounds(90,20,70,20);
    Label lb2=new Label(group1,SWT.NONE);
    lb2.setText("地址：");
    lb2.setBounds(10,50,70,20);
    Text text2=new Text(group1,SWT.BORDER);
    text2.setBounds(90,50,70,20);
}
TabItem tabItem2=new TabItem(tabFolder,SWT.NONE); //声明第2个选项页
tabItem2.setText("选项2");
{
     //创建第2个分组框，建立在tabFolder上
    Group group2=new Group(tabFolder,SWT.NONE);
    tabItem2.setControl(group2);    //让tabItem2控制group2
    group2.setText("兴趣爱好");
    Button bt1=new Button(group2,SWT.CHECK);
    bt1.setBounds(20,20,70,20);
    bt1.setText("音乐");
    Button bt2=new Button(group2,SWT.CHECK);
    bt2.setBounds(20,50,70,20);
    bt2.setText("美术");
    Button bt3=new Button(group2,SWT.CHECK);
    bt3.setBounds(20,80,70,20);
    bt3.setText("体育");
}
shell.pack();
shell.open();
while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
    if(!display.readAndDispatch()){  //如果display不忙
    display.sleep();    //休眠
```

```
            }
        }
        display.dispose();        //销毁display
    }
}
```

运行结果如图 4.13、4.14 所示。

图 4.13 选项 1

图 4.14 选项 2

4.4 布局管理器

在 Java 中， GUI 程序开发的目标之一是跨平台，而每种类型操作系统对屏幕的定义不一样，所以 Swing 中引入了布局的概念，对子组件的位置和大小等信息进行定义。 SWT 中也采用了布局方式，用户可使用布局来控制组件中元素的位置和大小等信息。

组件可以用方法 setBounds (int x, int y, int width, int height) 来指定该组件相对于父组件的位置和组件的大小。组件的这种定位方式称为绝对定位。当组件数量较多，布局较复杂时，则要使用布局管理器 LayoutManager 来进行定位，这时，每个控件的坐标 X、Y、宽度和高度都是通过 LayoutManager 设置的，这种定位方式称为托管定位。SWT 提供了一些常用的布局管理器供用户使用；在本章中，将介绍四种基本的布局管理器： FillLayout 、RowLayout 、 GridLayout 和 FormLayout 。在布局管理器中，每当重新设置复合组件的大小，都需要进行定位。

布局管理器常常是专为某一个复合组件设计的。一些布局管理器只使用它们自身的参数就可以控制，而另一些布局管理器还需要其它参数（ LayoutData ），该参数是在设置布局管理器的复合组件中的每个控件上指定的。 SWT 中常用的布局管理器有如下一些：

FillLayout ： 充满式布局，在容器中以相同的大小以单行或单列排列组件。

RowLayout ： 行列式布局，以单行或多行的方式定制组件的排列方式。

GridLayout ： 网格式布局，以网格的方式进行布局，组件可以占用指定的一个或几个网格。

FormLayout ： 表格式布局，通过定义组件四个边的距离来排列组件，被引用的相对的组件可以是父组件，也可以是同一容器中的其它组件。

4.4.1 充满式布局

充满式布局（FillLayout 类）是最简单的布局管理器。它把组件按一行或一列充满整个容器，并强制组件的大小一致。一般，组件的高度与最高组件相同，宽度与最宽组件相同。FillLayout 不能折行，不能设置边界距离和间距。如果容器中只有一个组件，则该组件会充满整个容器。

1．构造方法：

FillLayout()　创建按一行充满容器的对象。

FillLayout(int type)　创建按指定类型充满容器的对象，指定类型（type）有：

SWT.HORIZONTAL　按一行充满容器。

SWT.VERTICAL　按一列充满容器。

2．常用属性：

int type　指定组件充满容器的类型。type 的取值同上。

要将组件按一列充满容器，可以设置 type 属性，代码如下：

```
FillLayout filllayout=new FillLayout();  //创建FillLayout对象
Filllayout.type=SWT.VERTICAL;     //设置type的值
shell.setLayout(filllayout);  //将FillLayout对象用于shell上
new Button(shell,SWT.PUSH).setText("超宽按钮1");//在shell中创建按钮
new Button(shell,SWT.PUSH).setText("按钮2");
new Button(shell,SWT.PUSH).setText("按钮3");
new Button(shell,SWT.PUSH).setText("按钮4");
```

例 4.11 充满式布局示例。

```
package edu.ch4;
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
public class Sample4_11 {
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        final Shell shell=new Shell(display);//shell是程序的主窗体
        shell.setText("FillLayout示例");
        FillLayout filllayout=new FillLayout();  //创建FillLayout对象
        shell.setLayout(filllayout);  //将FillLayout对象用于shell上
        new Button(shell,SWT.PUSH).setText("超宽按钮1");//在shell中创建按钮
        new Button(shell,SWT.PUSH).setText("按钮2");
        new Button(shell,SWT.PUSH).setText("按钮3");
        new Button(shell,SWT.PUSH).setText("按钮4");
        shell.pack();
        shell.open();
        while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){  //如果display不忙
            display.sleep();    //休眠
            }
        }
        display.dispose();      //销毁display
    }
}
```

运行结果如图 4.15 所示。

图 4.15 FillLayout 水平布局

图 4.16 FillLayout 垂直布局

如果要将按钮按竖直方向排列，也可以只修改以下一行语句：

```
Layout layout=new FillLayout(SWT.VERTICAL);
```

运行结果如图 4.16 所示。

4.4.2 行列式布局

行列式布局（RowLayout 类）可以使组件折行显示，可以设置边界距离和间距。另外，还可以对每个组件通过 setLayoutData()方法设置 RowData 对象。RowData 用来设置组件的大小。

1．构造方法：

RowLayout()　创建按行放置组件的对象。

RowLayout(int type)　创建按指定类型放置组件的对象。指定类型（type）有：

SWT.VERTICAL　按列放置组件。

SWT.HORIZONTAL　按行放置组件。

2．常用属性：

int marginWidth：组件距容器边缘的宽度（像素），默认值为 0。

int marginHeight：组件距容器边缘的高度（像素），默认值为 0。

int marginTop：组件距容器上边缘的距离（像素），默认值为 3。

int marginBottom：组件距容器下边缘的距离（像素），默认值为 3。

int spacing：组件之间的距离，默认值为 3。

boolean justify：如果该属性为 true，则组件间的距离随容器的拉伸而变大。默认值为 false。

boolean wrap：如果该属性为 true，则当容器空间不足时会自动折行；如果该属性为 false，不自动折行。默认值为 true。

boolean pack：如果该属性为 true，组件大小为设定值；如果该属性为 false，则强制组件的大小相同。默认值为 true。

int type：使组件按指定式样放置，（type=SWT.HORIZONTAL|SWT.VERTICAL），默认为按行放置，默认值为 SWT.HORIZONTAL。

3．RowData 类：

RowData 称为 RowLayout 的布局数据类，可用于改变容器中组件的外观形状。其构造方法：

RowData(int width,int height);

例如：

```
Button bt1=new Button(shell,SWT.PUSH); //创建按钮
bt1.setText("按钮1");
RowData rowdata=new RowData(60,30); //创建布局数据类的对象
bt1.setLayoutData(rowdata);              //设置按钮的布局数据
```

利用 RowData 对象设置按钮（bt1）的宽度是 60 像素，高度是 30 像素。

例 4.12 行列式布局。

```
package edu.ch4;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
public class Sample4_12 {
    public static void main(String[] args) {
        Display display=new Display();//创建一个display对象。
        final Shell shell=new Shell(display);//shell是程序的主窗体
        shell.setText("FillLayout示例");
        RowLayout rowlayout=new RowLayout();  //创建按行放置组件的对象
        rowlayout.pack=false;        //强制组件大小相同
        rowlayout.wrap=false;        //不自动折行
        rowlayout.marginWidth=20;     //组件距容器边缘的宽度为20像素
        rowlayout.marginHeight=20;    //组件距容器边缘的高度为20像素
        rowlayout.spacing=10;         //组件之间的间距为10像素
        shell.setLayout(rowlayout);  //设置容器shell的布局方式为rowlayout
        Button bt1=new Button(shell,SWT.PUSH); //创建按钮
        bt1.setText("按钮1");
        RowData rowdata=new RowData(80,30); //创建布局数据类的对象
        bt1.setLayoutData(rowdata);          //设置按钮的布局数据
        new Button(shell,SWT.PUSH).setText("按钮2");
        new Button(shell,SWT.PUSH).setText("按钮3");
        new Button(shell,SWT.PUSH).setText("按钮4");
        shell.pack();        //自动调整容器shell的大小
        shell.open();        //打开主窗体
        while(!shell.isDisposed()){  //如果主窗体没有关闭则一直循环
            if(!display.readAndDispatch()){  //如果display不忙
            display.sleep();     //休眠
            }
        }
        display.dispose();       //销毁display
    }
}
```

当 rowlayout.pack=true 时，各按钮为设定值，如图 4.17 所示；当 rowlayout.pack=false 时，各按钮被强制设定为相同大小，如图 4.18 所示；当 rowlayout.justify=true 时，将主窗体拉伸，各按钮间的距离也增大，但间隔均匀分布，如图 4.19 所示；当 rowlayout.justify=false 时，将主窗体拉伸，各按钮间距不变，如图 4.20 所示；当 rowlayout.wrap=false 时，当主窗体宽度收缩，按钮自动折行，如图 4.21 所示；当 rowlayout.wrap=false 时，主窗体宽度收缩，按钮不会折行，如图 4.22 所示。

图 4.17　rowlayout.pack=true

图 4.18 rowlayout.pack=false

图 4.19 rowlayout.justify=true

图 4.20 rowlayout.justify=false

图 4.21 rowlayout.wrap=true

图 4.22 rowlayout.wrap=false

4.4.3 网格式布局

网格式布局（GridLayout 类）是实用而且功能强大的标准布局，也是较为复杂的一种布局。这种布局把容器分成网格，把组件放置在网格中。GridLayout 有很多可配置的属性，和 RowLayout 一样，也有专用的布局数据类 GridData， GridLayout 的强大之处在于它可以通过 GridData 来设置每一个组件的外观形状。GridLayout 的构造方法无参数，但可以通过 GfidData 和设置 GridLayout 的属性来设置组件的排列及组件的形状和位置。

1．GridLayout 的属性

int numColumns：设置容器的列数，组件从左到右按列放置，当组件数大于列数时，下一个组件将自动添加新的一行。默认值为 1 列。

boolean makeColumnsEqualWidth：强制使列都具有相同的宽度，默认值为 false。

int marginWidth：设置组件与容器边缘的水平距离，默认值为 5。

int marginHeight：设置组件与容器边缘的垂直距离，默认值为 5。

int horizontalSpacing：设置列与列之间的间隔，默认值为 5。

int verticalSpacing：设置行与行之间的间隔，默认值为 5。

例 4.13 GridLayout 的属性设置。

```
package edu.ch4;
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
public class Sample4_13 {
    public static void main(String[] args) {
        Display display = new Display();
         Shell shell = new Shell(display);
         shell.setText("GridLayout示例");
         GridLayout gridLayout = new GridLayout();
         gridLayout.numColumns = 3;
         gridLayout.horizontalSpacing=30;
         gridLayout.makeColumnsEqualWidth=true;
         shell.setLayout(gridLayout);
```

```
            new Button(shell, SWT.PUSH).setText("B1");
            new Button(shell, SWT.PUSH).setText("超宽按钮 2");
            new Button(shell, SWT.PUSH).setText("按钮 3");
            new Button(shell, SWT.PUSH).setText("B4");
            new Button(shell, SWT.PUSH).setText("按钮 5");
            shell.pack();
            shell.open();
            while (!shell.isDisposed()) {
                if (!display.readAndDispatch()) display.sleep();
            }
            display.dispose();
        }
}
```

当 gridLayout.numColumns 分别为 1、2、3 时，按钮依次按 1 列、2 列和 3 列排列，运行结果如图 4.23~4.25 所示；当 makeColumnsEqualWidth=true 时，虽然按钮宽度不同，但列宽相同，如图 4.26 所示；当 horizontalSpacing=30 时，列间距为 30，如图 4.27 所示。

图 4.23 numColumns = 1

图 4.24 numColumns = 2

图 4.25 numColumns = 3

图 4.26 makeColumnsEqualWidth=true

图 4.27 horizontalSpacing=30

2．布局数据类（GridData 类）

GridData 是 GridLayout 专用的布局数据类，用 GridData 可以构建很多复杂的布局方式。

① GridData 的构造方法如下：

GridData(); 创建一个属性值为默认值的对象。

GridData(int type); 创建一个指定类型（type）的对象。

② GridData 常用类型如下：

GridData.FILL 通常与 GridData 类的对象属性 horizontalAlignment 和 verticalAlignment 配合使用，充满对象属性指定的空间。

GridData. FILL_HORIZONTAL 水平充满，组件充满网格水平方向的空间。

GridData. FILL_VERTICAL 垂直充满，组件充满网格垂直方向的空间。

GridData. FILL_BOTH 双向充满，组件充满水平和垂直方向的空间。

GridData.HORIZONTAL_ALIGN_BEGINNING 水平对齐靠左，组件在网格中靠左放置。

GridData.HORIZONTAL_ALIGN_CENTER 水平对齐居中，组件在网格中居中放置。

GridData.HORIZONTAL_ALIGN_END 水平对齐靠右，组件在网格中靠右放置。

③ GridData 常用对象属性如下：

int horizontalSpan　设置组件占用的列数，默认值为 1。

int verticalSpan　设置组件占用的行数，默认值为 1。

horizontalAlignment　设置组件的对齐方式为水平方向。

verticalAlignment　设置组件的对齐方式为垂直方向。

grabExcessHorizontalSpace　抢占额外的水平空间。

grabExcessVerticalSpace　抢占额外的垂直空间。

horizontalAlignment 和 verticalAlignment 可以取以下值：

BEGINNING　开始（水平对齐时居左；垂直对齐时居上）

CENTER　居中

END　结束（水平对齐时居右；垂直对齐时居下）

FILL　充满

默认的 horizontalAlignment 值是 BEGINNING。默认的 verticalAlignment 值是 CENTER。

例 4.14 使用 gridData 布局。

```
package edu.ch4;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
public class Sample4_14 {
    public static void main(String[] args) {
        Display display = new Display();
         Shell shell = new Shell(display);
         shell.setText("GridData示例");
         GridLayout gridLayout = new GridLayout();//创建网格布局对象
         gridLayout.numColumns = 3;       //设置网格布局列数为3
         gridLayout.makeColumnsEqualWidth=true;  //强制列宽相等
         shell.setLayout(gridLayout);        //将shell设置为指定的网格布局式样
         GridData gridData=new GridData();    //创建网格布局数据对象
         gridData.horizontalSpan = 2;     //水平方向跨2列
         gridData.verticalSpan=2;         //垂直方向跨2行
         gridData.horizontalAlignment = GridData.CENTER;  //水平方向居中
         gridData.verticalAlignment = GridData.FILL; //垂直方向充满
         Button b1=new Button(shell, SWT.PUSH); //创建按钮对象b1
         b1.setText("B1");
         b1.setLayoutData(gridData);     //将设定的网格布局数据用于按钮对象b1
         new Button(shell, SWT.PUSH).setText("超宽按钮 2");
         new Button(shell, SWT.PUSH).setText("按钮 3");
         Button b4=new Button(shell, SWT.PUSH);
         b4.setText("B4");
         //用带参数的构造方法创建gridData对象
         gridData = new GridData(GridData.FILL_HORIZONTAL);
         b4.setLayoutData(gridData);       //将gridData用于b4，水平方向充满
         Button b5=new Button(shell, SWT.PUSH);
         b5.setText("按钮 5");
```

```
            gridData = new GridData();
            gridData.horizontalAlignment = GridData.FILL;//设置b5为水平方向充满
            b5.setLayoutData(gridData);
            new Button(shell, SWT.PUSH).setText("按钮 6");
            Text t1=new Text(shell,SWT.BORDER);
            t1.setText("文本框 1");
            gridData = new GridData();
            gridData.verticalSpan = 2;    //跨两行
            gridData.horizontalSpan=2;    //跨两列
            gridData.verticalAlignment = GridData.FILL; //垂直方向充满
            gridData.grabExcessVerticalSpace = true;  //抢占垂直方向额外空间
            gridData.horizontalAlignment = GridData.FILL;//水平方向充满
            gridData.grabExcessHorizontalSpace = true;//抢占水平方向额外空间
            t1.setLayoutData(gridData);              //gridData用于文本框t1
            new Button(shell, SWT.PUSH).setText("按钮 7");
            new Button(shell, SWT.PUSH).setText("按钮 8");
            shell.pack();
            shell.open();
            while (!shell.isDisposed()) {
                if (!display.readAndDispatch()) display.sleep();
            }
            display.dispose();
        }
}
```

运行结果如图 4.28 所示。虽然按钮 b1 和文本框 t1 都占两行两列，但由于 t1 水平方向和垂直方向的 alignment 属性都是 FILL，因此文本框 t1 充满了两行和两列的空间，而按钮 b1 的 horizontalalignment 属性是 CENTER，而 verticalalignment 属性是 FILL，所以水平方向居中放置，而垂直方向充满了两行。按钮 b4 和 b5 采用了不同的构造方法来创建 gridData 对象，但都充满了该列的水平空间。

当窗体变大时，由于设置了抢占水平方向和垂直方向额外的空间，即 grabExcessVerticalSpace = true 和 grabExcessHorizontalSpace = true，所以文本框 t1 随窗体的拉伸而变大，反之，当窗体缩小时，t1 也会缩小。按钮 b4 和 b5 设置了水平方向充满属性，所以窗体拉伸时水平方向也会随之拉伸。其余的组件大小不变。如图 4.29 所示。这说明，如果组件所在的行变宽或列变高，所有具有填充（FILL）属性的组件也会变宽或变高；而具有 BEGINNING、CENTER、END 属性的组件不会改变其大小。

图 4.28 GridData 示例

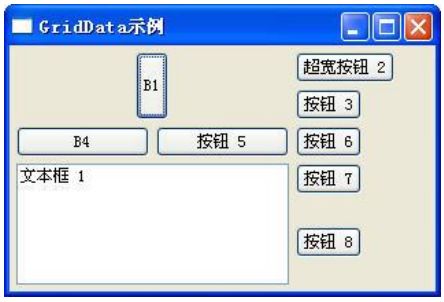

图 4.29 窗体拉伸时 b4、b5 和 t1 变化

4.4.4 表格式布局

表格式布局（FormLayout 类）是一种非常灵活、精确的布局方式，这个布局是 SWT2.0 版新增的。FormLayout 也有专用的布局数据类 FormData，此外，还增加了一个 FormAttachment 类。FormAttachment 定义了组件的四边与父容器（Shell、Composite 等）的边距，为保证组件在父容器中的相对位置不变，FormAttachment 类用不同的构造方法来实现组件的定位，用 FormData 和 FormAttachment 配合，可以创建复杂的界面，而且当主窗体大小改变时，组件的相对位置能保持相对不变。FormLayout 的构造方法：FormLayout()。

1．FormLayout 的属性

int marginWidth：设置组件与容器边缘的水平距离，默认值为 0。

int marginHeight：设置组件与容器边缘的垂直距离，默认值为 0。

例如，以下代码把父容器（shell）的四周边距都设置成 10 像素。

```
Display display = new Display ();
Shell shell = new Shell (display);
FormLayout formlayout= new FormLayout ();
formlayout.marginHeight = 10;
formlayout.marginWidth = 10;
shell.setLayout (formlayout);
```

2．FormData 类

① FormData 的构造方法

FormData()　默认构造方法，组件的宽度和高度要用属性 width 和 height 设置。

FormData(int width,int height)　参数 width 和 height 设置组件的宽度和高度。

② FormData 的属性

width　设置组件的宽度。

height　设置组件的高度。

top　和 FormAttachment 配合设置组件顶部和父容器顶部的边距。

bottom　和 FormAttachment 配合设置组件底部和父容器底部的边距。

left　和 FormAttachment 配合设置组件左边和父容器左边的边距。

right　和 FormAttachment 配合设置组件右边和父容器右边的边距。

如果 FormData 中的 width 和 height 设置的宽度和高度与 FormAttachment 设置的约束发生冲突，则按照 FormAttachment 设置，width 和 height 的设定值就不起作用了。

3．FormAttachment 类

Attachment 的含义是附着、粘贴。FormAttachment 类就是用来指定组件在父容器中的粘贴位置。FormAttachment 计算组件粘贴位置和组件大小的方法是依据下面的表达式：

y = ax + b

表达式中 y 是纵坐标，从上往下是正方向；x 是横坐标，从左至右是正方向；a 是斜率（a=m/n，n≠0），b 是偏移量，沿 x、y 轴正方向的偏移量为正，反之为负。.

① FormAttachment 的构造方法

FormAttachment()　组件紧贴父容器的左边缘和上边缘，如果父容器设置了 FormLayout 属性 marginWidth 和 marginHeight，则距父容器的上边缘和左边缘为 marginHeight 和 marginWidth 的设定值。

FormAttachment(Control control)　以指定的组件 control 为参照物。

FormAttachment(Control control, int offset)　以指定的组件 control 为参照物，相对指定

组件的偏移量为 offset。

FormAttachment(Control control, int offset,int alignment) 以指定的组件 control 为参照物，相对指定组件的偏移量为 offset，对齐方式为 alignment。alignment 的取值如下：

SWT.TOP、SWT.BOTTOM、SWT.LEFT、SWT.RIGHT、SWT.CENTER

FormAttachment(int m,int n, int offset) 以组件相对于父容器宽度或高度的百分比（即斜率 a）来给组件定位，m 为 a 的分子，n 为 a 的分母，offset 是偏移量。

FormAttachment(int m, int offset) 以组件相对于父容器宽度或高度的百分比（即斜率 a）来给组件定位，m 为 a 的分子，a 的分母为默认值 100，offset 是偏移量。

FormAttachment(int m) 以组件相对于父容器宽度或高度的百分比（即斜率 a）来给组件定位，m 为 a 的分子，a 的分母为默认值 100，偏移量为默认值 0。

例 4.15 FormData 与 FormAttachment 的配合使用。

```
package edu.ch4;
import org.eclipse.swt.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
public class Sample4_15 {
     public static void main(String[] args) {

    Display display = new Display ();
    Shell shell = new Shell (display);
    shell.setText("FormLayout示例");

    FormLayout formLayout= new FormLayout();  //创建表格布局对象formLayout
    shell.setLayout(formLayout);         //设置shell的布局方式为表格布局
    formLayout.marginHeight = 10;        //设置shell的上下边距为10像素
    formLayout.marginWidth = 20;         //设置shell的左右边距为10像素
    Button b1=new Button(shell,SWT.PUSH);
    b1.setText("B1");
    FormData formData1=new FormData();  //创建布局数据对象formData1
    formData1.width=100;      //按钮b1的宽度为100像素
    formData1.height=50;      //按钮b1的高度为50像素
    b1.setLayoutData(formData1);  //设置b1的布局数据为formData1

    Button b2=new Button(shell,SWT.PUSH);
    b2.setText("B2");
    //创建FormAttachment对象formAttachment,以b1为参照物
    FormAttachment formAttachment=new FormAttachment(b1); //指定B1为参照物
    FormData formData2=new FormData(50,30);  //创建FormData对象，宽度50，高度30
    formData2.left=formAttachment;  //b2的左边紧贴与b1的右边
    b2.setLayoutData(formData2);     //设置b2的布局数据为formData2

    Button b3=new Button(shell,SWT.PUSH);
    b3.setText("B3");
    FormData formData3=new FormData();//创建布局数据对象formData3
```

```
        formData3.top=new FormAttachment(b2,10,SWT.BOTTOM);//b2的底边与b3的顶部距离为10
        formData3.left=new FormAttachment(b2,0,SWT.LEFT);//b2的左边与b3左边位移为0，
                                                          //即左边对齐
        formData3.right=new FormAttachment(b2,0,SWT.RIGHT);  //b2的右边与b3右边对齐
        b3.setLayoutData(formData3); //设置b3的布局数据为formData3

        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
          if (!display.readAndDispatch()) {
            display.sleep();
          }
        }
        display.dispose();
      }
    }
```

运行结果如图 4.30 所示。本例中，按钮 B1 的宽度和高度由 FormData 设置，按钮的位置为默认的位置，即 shell 的左上角，边距由 maginHeight 和 maginWidth 设置。按钮 B2 以 B1 为参照物，位置紧靠 B1 右侧，B2 的上部与 shell 的距离由 maginHeight 设定，B2 的右边与 shell 的边距由 maginWidth 设定。按钮 B3 以 B2 为参照物，FormData3.TOP 设定了 B3 的顶部距离 B2 的底部位移为 10 像素、FormData3.LEFT 设定了 B3 与 B2 左边对齐、FormData3.RIGHT 设定了 B3 与 B2 右边对齐。如果要使 B1 右侧与 B2 左侧相距 20 相素，可将 FormAttachment 的构造方法增加偏移量 20，如：

```
FormAttachment formAttachment=new FormAttachment(b1,20);
```

运行结果如图 4.31 所示。

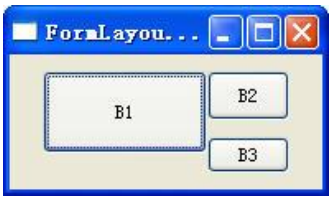

图 4.30 FormAttachment(b1)

图 4.31 FormAttachment(b1,20)

构造方法 FormAttachment(int m,int n, int offset) 是以组件相对于父容器宽度或高度的百分比（即斜率 a）来给组件定位的，当分子 m=0 时，组件以父容器的左边或上边为基准，偏移量为正数；当分子 m=100 时，组件以父容器的右边或下边为基准，偏移量为负数。当父容器的大小改变时，组件与父容器的基准边保持设定的距离，组件的大小随父容器的改变而变化。

例 4.16 构造方法 FormAttachment(int m,int n, int offset)的使用示例。

```
package edu.ch4;
import org.eclipse.swt.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
public class Sample4_16 {
    public static void main(String[] args) {
        Display display = new Display ();
```

```
        Shell shell = new Shell (display);
        shell.setText("FormLayout示例");

        FormLayout formlayout= new FormLayout();
        formlayout.marginHeight=10;      //设置shell的上、下边缘和组件的距离为10像素
        Button b1=new Button(shell,SWT.PUSH);
        b1.setText("B1");
        FormData formData1=new FormData();  //创建布局数据对象formData1
        formData1.top=new FormAttachment(0,50);       //设置组件B1的顶部离父容器
                                                      //shell上边缘的距离为50像素
        formData1.bottom=new FormAttachment(100,-50);//设置组件B1的底部离shell的下
                                                      //边缘的距离为50像素
        formData1.left=new FormAttachment(0,50);      //设置组件B1的左边离shell的左
                                                      //边距离为50像素
        formData1.right=new FormAttachment(100,-50); //设置组件B1的右边离shell的右
                                                      //边距离为50像素
        formData1.width=100;      //按钮b1的宽度为100像素
        formData1.height=50;      //按钮b1的高度为50像素
        b1.setLayoutData(formData1);  //设置b1的布局数据为formData1
        Button b2=new Button(shell,SWT.PUSH);
        b2.setText("B2");
        FormAttachment formAttachment=new FormAttachment();//创建FormAttachment对象
        FormData formData2=new FormData(50,30);  //创建FormData对象，宽度50，高度30
        formData2.left=formAttachment;  //B2的左边与shell左边缘的距离为0
        formData2.top=formAttachment;    //B2的上边与shell上边缘的距离为
                                        //marginHeight设定的值（10像素）
        b2.setLayoutData(formData2);     //设置b2的布局数据为formData2
        shell.setLayout(formlayout);
        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
          if (!display.readAndDispatch()) {
            display.sleep();
          }
        }
        display.dispose();
    }
}
```

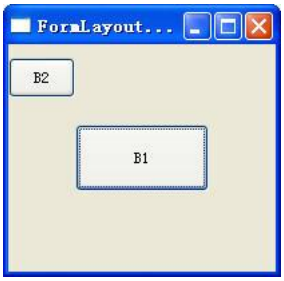

图 4.32 B1 与 shell 四边的距离相等

图 4.33 shell 改变后 B1 与其四边的距离仍相等

例 4.17 综合布局示例。

```
package edu.ch4;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.SWT;
public class Sample4_17 {
  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setText("FormLayout示例");
    FormLayout formlayout = new FormLayout(); //创建表格布局对象
    shell.setLayout(formlayout);
    Label label=new Label(shell,SWT.BORDER);  //在shell中创建Label对象
    label.setText("Label One");
    FormData data = new FormData();
    data.top = new FormAttachment(0, 5);  //Label与shell上边框相距5象素
    data.left = new FormAttachment(0, 5); // Label与shell左边框相距5象素
    data.bottom = new FormAttachment(50, -5);//Label在shell水平中线上方5象素
    data.right = new FormAttachment(50, -5); // Label在shell垂直中线左边5象素
    label.setLayoutData(data);
    Composite composite = new Composite(shell, SWT.NONE);//创建面板
    GridLayout gridLayout = new GridLayout();  //创建网格布局对象
    gridLayout.marginHeight = 0;
    gridLayout.marginWidth = 0;
    composite.setLayout(gridLayout);    //设置面板布局方式为网格布局
    Button b1 = new Button(composite, SWT.PUSH); //在composite上创建button B1
    b1.setText("B1");
    GridData gridData = new GridData(GridData.FILL_BOTH); //设置布局方式为双向填充
    b1.setLayoutData(gridData);
    Button b2 = new Button(composite, SWT.PUSH);  //B2设置同B1
    b2.setText("B2");
    gridData = new GridData(GridData.FILL_BOTH);
    b2.setLayoutData(gridData);
    Button b3 = new Button(composite, SWT.PUSH); //B2设置同B1
    b3.setText("B3");
    gridData = new GridData(GridData.FILL_BOTH);
```

```
    b3.setLayoutData(gridData);
    data = new FormData();     //创建FormData对象
    data.top = new FormAttachment(0, 5);      //设置composite距shell上边框5象素
    data.left = new FormAttachment(label, 5);//设置composite距label 5象素
    data.bottom = new FormAttachment(50, -5);//设置composite在shell水平中线上方5象素
    data.right = new FormAttachment(100, -5);//设置composite在shell右边框的左侧5象素
    composite.setLayoutData(data);            //设置composite的布局数据
    Text text=new Text(shell,SWT.BORDER);    //创建Text对象
    text.setText("Text");
    data = new FormData();     //创建表格布局数据
    data.top = new FormAttachment(label, 5);    //text上方离label 5象素
    data.left = new FormAttachment(0, 5);       // text左边离shell左边框5象素
    data.bottom = new FormAttachment(100, -5); // text下边框离shell下边框5象素
    data.right = new FormAttachment(100, -5);  // text右边框离shell右边框5象素
    text.setLayoutData(data);                   //设置text的布局数据
    shell.pack();
    shell.open();
    while (!shell.isDisposed()) {
      if (!display.readAndDispatch()) {
        display.sleep();
      }
    }
    display.dispose();
  }
}
```

运行结果如图 4.34 所示。当窗体 shell 大小改变时，各组件的相对位置保持不变。如图 4.35 所示。

图 4.34 综合实例

图 4.35 窗体变化时

4.5 SWT 的事件处理

SWT 的事件模型和 Java 标准的 AWT 基本相同。事件产生处的 SWT 组件称为事件源，对事件作出具体动作称为监听器（Listener）。监听器负责监听组件上的事件，并对发生的事件进行处理。基本的模式是将一个监听器添加到已经创建的组件中，当相应的事件发生时，监听器的代码就会被执行。

4.5.1 SWT 的常用事件

每一种类型的监听器，都有一个接口来定义这种监听器，由类提供事件信息，由应用程序接口方法负责添加监听器。如果一个监听器接口中定义了多个方法，则会提供一个适配器来实现监听器接口并同时提供空方法。所有的事件、监听器和适配器都放在包 **org.eclipse.swt.events** 中。

例如，添加组件选择事件的监听器为 **addSelectionListener**，事件为 **SelectionEvent**，相应的适配器为 **SelectionAdapter**。添加鼠标事件的监听器为 **addMouseListener**，事件为 **MouseEvent**，相应的适配器为 **MouseAdapter**。**SWT** 中常用的事件如下：

1．addMouseListener 鼠标监听器。常用方法：

mouseDown() 鼠标按下时触发。

mouseUP() 鼠标放开时触发。

mouseDoubleClick() 鼠标双击时触发。

2．addKeyListener 按键监听器。常用方法：

keyPressed() 当焦点在组件上时，按下键盘任一键时触发。但对某些组件（如按钮 **Button**），按回车键时不能触发。

keyReleased() 按键弹起时触发。

3．addSelectionListener 组件选择监听器。常用方法：

widgetSelected() 当组件被选择（单击鼠标、焦点在组件上时按回车键）时触发。

4．addFocusListener 焦点监听器。常用方法：

focusGained() 得到焦点时触发。

focusLost() 失去焦点时触发。

4.5.2 SWT 的常用监听器应用实例

例 4.18 鼠标监听器，监听鼠标双击事件。

```
package edu.ch4;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.SWT;
public class Sample4_18 {
    static Text text;
    public static void main(String[] args) {
            Display display = new Display();
          final  Shell shell = new Shell(display);
            RowLayout rowLayout=new RowLayout();
            rowLayout.marginWidth=20;
            rowLayout.marginHeight=30;
            shell.setLayout(rowLayout);
            shell.setText("SWT事件处理示例");
```

```
            text=new Text(shell,SWT.BORDER|SWT.WRAP);
           RowData rowData=new RowData();
              rowData.width=100;
              rowData.height=50;
          text.setLayoutData(rowData);
           //将鼠标监听器用于text组件
          text.addMouseListener(new MouseAdapter() {    //采用鼠标监听适配器
          public void mouseDoubleClick(MouseEvent e) { //监听鼠标双击事件的方法
              text.setText("文本框中鼠标双击事件发生！");  //在text中显示信息
              //声明信息对话框对象，并在对话框中显示信息
             MessageBox dialog=new MessageBox(shell,SWT.OK|SWT.ICON_INFORMATION);
                    dialog.setText("Double click");
                    dialog.setMessage("文本框中鼠标双击事件发生！");
                    dialog.open();
          }});
          shell.pack();
          shell.open();
          while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
              display.sleep();
            }
          }
          display.dispose();
    }
}
```

运行结果如图 4.36 所示。当在文本框中双击鼠标时，文本框中显示信息并弹出一个对话框，如图 4.37 所示。

图 4.36 鼠标双击事件

图 4.37 双击后弹出的对话框

例 4.19 键盘监听器，监听键盘事件。

```
package edu.ch4;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.*;
public class Sample4_19 {
      Text text1,text2;
```

```
    public Sample4_19() {
        Display display = new Display();
        Shell shell = new Shell(display,SWT.SHELL_TRIM);
        GridLayout layout=new GridLayout();
        layout.numColumns=2;
        shell.setLayout(layout);
        shell.setText("Event demo");
        Label label1=new Label(shell,SWT.RIGHT);
        label1.setText("text1:");
        text1=new Text(shell,SWT.BORDER|SWT.WRAP);
        GridData gridData1=new GridData(100,30);
        text1.setLayoutData(gridData1);
        text1.addKeyListener(new KeyAdapter(){    //添加按键监听器于text1上
           public void keyPressed(KeyEvent e) {    //监听键盘按键
            if(e.keyCode==SWT.CR)                  //当按键为回车键时触发
                text2.setText(text1.getText());}});
        Label label2=new Label(shell,SWT.RIGHT);
        label2.setText("text2:");
        text2=new Text(shell,SWT.BORDER|SWT.WRAP);
        GridData gridData2=new GridData(100,30);
        text2.setLayoutData(gridData2);
        text2.setEditable(false);
        text2.setBackground(new Color(display,255,255,255));
        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }
    public static void main(String[] args) {
        Sample4_19 s4_19=new Sample4_19();
    }
}
```

在文本框 text1 中输入信息，按回车键时将 text1 中的信息显示在 text2 中。运行结果如图 4.38 所示。

图 4.38 键盘监听事件

例 4.20 组件选择监听器，监听组件选择事件。

```
package edu.ch4;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.SWT;
public class Sample4_20 {
    static Display display = new Display();
    static  final  Shell shell = new Shell(display,SWT.SHELL_TRIM);
    public static void main(String[] args) {
        shell.setText("组件选择事件示例");
        Button button=new Button(shell,SWT.PUSH);
        button.setText("请点击我");
        RowLayout layout=new RowLayout();
        layout.marginHeight=10;
        layout.marginWidth=20;
        shell.setLayout(layout);
        RowData data=new RowData(80,40);
        button.setLayoutData(data);
        button.addSelectionListener(new SelectionAdapter(){
         public void widgetSelected(SelectionEvent e){
             MessageBox dialog=new MessageBox(shell,SWT.OK|SWT.ICON_INFORMATION);
                 dialog.setText("组件选择事件");
                 dialog.setMessage("你好，SWT世界！");
                 dialog.open();
                   }
        });
        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }
}
```

运行结果如图 4.39 所示，当点击按钮时，弹出一个信息对话框，如图 4.40 所示。

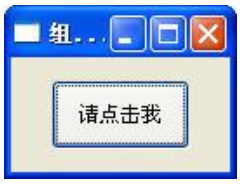

图 4.39 组件选择事件

图 4.40 信息对话框

以上示例中，在button.addSelectionListener()方法内部建立了一个匿名内部类，该类继承于SelectionAdapter，在widgetSelected()方法中写下了事件处理的代码。这种方式书写代码简单方便，容易掌握。但由于事件处理代码会随着组件一起分散在代码中的各个部分，当工具栏、菜单栏等也需要处理相同的用户行为时，无法重用事件中的处理代码。

事件处理的另一种方法是采用命名内部类。可以解决匿名内部类存在的问题。把例4.20改为命名内部类，代码如下：

```
package edu.ch4;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.SWT;
public class Sample4_20_2 {
    static Display display = new Display();
    static  final  Shell shell = new Shell(display,SWT.SHELL_TRIM);
    public static void main(String[] args) {
         shell.setText("组件选择事件示例");
         Button button=new Button(shell,SWT.PUSH);
         button.setText("请点击我");
         RowLayout layout=new RowLayout();
         layout.marginHeight=10;
         layout.marginWidth=20;
         shell.setLayout(layout);
         RowData data=new RowData(80,40);
         button.setLayoutData(data);
         button.addSelectionListener(new MySelectionListener());
         shell.pack();
         shell.open();
         while (!shell.isDisposed()) {
             if (!display.readAndDispatch()) {
                 display.sleep();
             }
         }
         display.dispose();
    }
    private static final class MySelectionListener extends SelectionAdapter{
        public void widgetSelected(SelectionEvent e){
         MessageBox dialog=new MessageBox(shell,SWT.OK|SWT.ICON_INFORMATION);
              dialog.setText("组件选择事件");
              dialog.setMessage("你好，SWT世界！");
              dialog.open();
                }
    }
}
```

运行结果相同。类 MySelectionListener 在主类 Sample4_20_2 的内部，所以称为命名内部

类。还可以将 MySelectionListener 类单独写成一个类文件，成为外部类。读者可以自己尝试一下。

4.6 SWT Designer 简介

SWT Designer 是一种功能强大且容易使用的基于 Eclipse SWT 技术的图形用户界面设计工具，是一个很好的 Eclipse 的界面开发插件包。利用 SWT Designer 的可视化界面，只需采用拖拉操作，就可以很快地在窗体上创建各种组件，设计出来的窗体和组件的外观和操作系统平台下其他软件的外观相似，具有本机系统的风格。SWT Designer 可以自动生成 Java 代码，利用它的属性编辑器还可以改变组件的各种属性，使 SWT 界面开发变得非常容易。

4.6.1 SWT Designer 的下载和安装

SWT Designer 有多个版本，要和相应的 Eclipse 版本相匹配。本书使用 Eclipse3.2，SWT Designer6.1.0。安装步骤如下：

1．下载

下载版本：SWT Designer6.1.0，文件名：Designer_v6.1.0_win32_x86.exe，下载地址：http://www.swt-designer.com/download_content.html

2．安装

在 Windows 下双击文件名，可自动安装在指定的目录下，且自动在 Eclipse 的 links 目录下生成 com.instantiations.designer.link 文件。安装结束后，启动 Eclipse，在主菜单中选择【文件】→【新建】→【项目】，如果在弹出的“新建项目”对话框中多了一个“Designer”项，则表示 SWT Designer 安装成功。如图 4.41 所示。

3．注册

在 Eclipse 主菜单中选择【窗口】→【首选项】，弹出一个对话框，如图 4.42，单击左边树形目录中的 Designer，然后单击对话框右下方的【Registration and Activation】按钮，弹出对话框，如图 4.43 所示，在 Serial number 栏中输入产品序列号，在 Activation Key 栏中输入产品激活号，单击【完成】，即可激活。SWT Designer 有免费版、评估版和正式版，免费版只有有限的功能，评估版需要填写用户资料和 E-mail 地址，从 E-mail 中可以获得 Activation Key，评估时限为 2 周。正式版则要购买产品，才能获得 Activation Key。

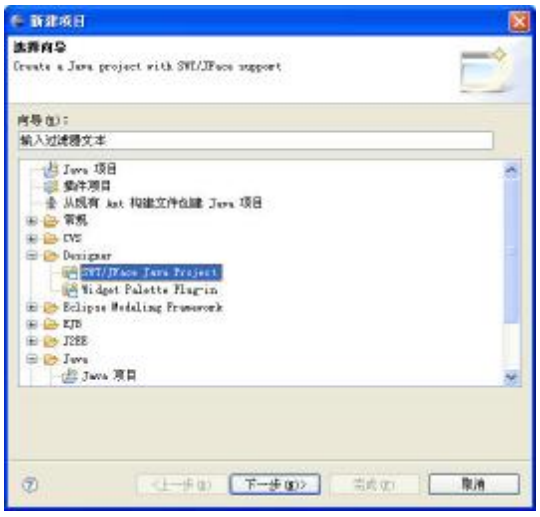

图 4.41 新建项目

图 4.42

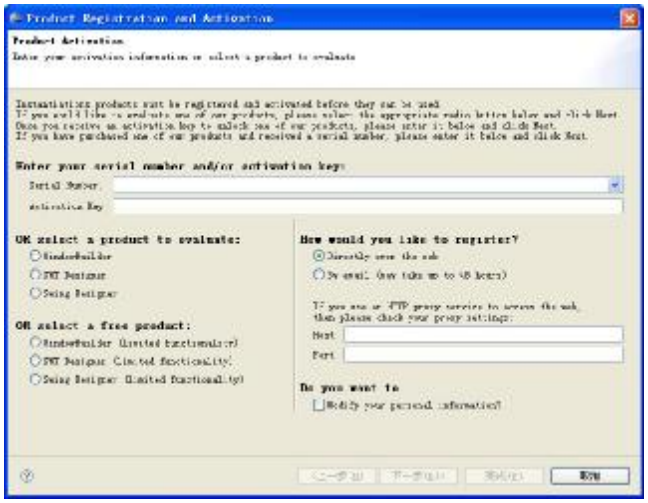

图 4.43

4.6.2 SWT Designer 开发实例

本节通过创建一个项目，熟悉一下 SWT Designer 的主界面及其相应的组件。操作步骤如下：

①创建 SWT/JFace 项目

在 Eclipse 主菜单中选择【文件】→【新建】→【其他】，弹出“新建”对话框，展开 Designer 节点，选择“SWT/JFace Java Project”，再单击【下一步】，出现“创建 Java 项目”对话框，输入项目名，如 SWTDesigerTest，单击【完成】。

②创建 Application Window 窗体

右单击项目名（SWTDesigerTest），选择【新建】→【其他】，在“新建”对话框中，展开 Designer 节点下的 SWT 节点，选择“Application Window”，再单击【下一步】，在弹出的对话框中输入包名和类名，在下方的单选项中选择“public static main() mathod”，该选项会自动生成 main()方法，使窗体能独立运行。单击【完成】。在程序编辑区的下方，有【Source】和【Design】两个标签，选择【Source】标签，则在程序编辑区中显示源程序；

选择【Design】标签，则在程序编辑区中出现一个窗体。

③在窗体中加入组件

加入 2 个标签组件（Label）、2 个文本框组件(Text)和 2 个按钮组件（Button），如图 4.44 所示。

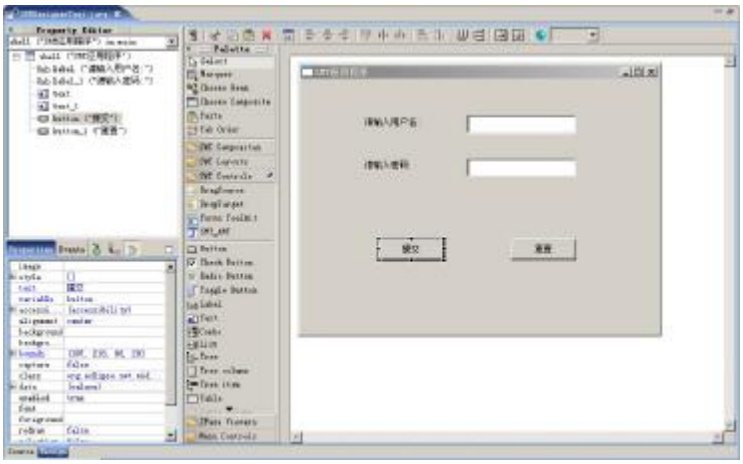

图 4.44 SWT Designer 界面

图 4.44 是 SWT Designer 完整的界面。该界面的左侧是属性编辑器（Property Editor），属性编辑器又分为上、下两部分，上面是一个树形目录，显示了组件间的继承关系。下面有两个标签，一个是属性（Properties）标签，一个是事件（Events）标签。选中属性（Properties）标签可打开属性页，可以编辑当前组件的各类属性。如图中当前组件是一个按钮(Button)，在属性页中把 text 属性改为“提交”。还可以改变组件显示的文字的字体（font）及其他属性。

④添加事件处理代码

在图 4.44 中的属性页中，选中事件（Events）标签可打开事件页，如图 4.45 所示，可以添加各种事件。选择需要添加的事件，在右侧空格双击，可自动生成相应的事件处理方法。如图中添加了组件选择事件（widgetSelected），相应的事件处理方法自动生成在源程序的第 48 行，可以在方法中加入事件处理代码。也可以在窗体中双击需要生成事件处理方法的组件，如本例中在窗体中双击“提交”按钮，会自动生成 widgetSelected()方法，并打开源程序，光标停留在该方法的头部。在方法体中加入以下代码：

```
if(text.getText()!=""|text_1.getText()!="")
//调用JFace的信息对话框显示登录信息
MessageDialog.openInformation(shell,"登录信息","欢迎"+text.getText()+"进入系统！");
        else
        //调用JFace的错误对话框显示出错信息
        MessageDialog.openError(shell,"错误","用户名或密码为空，请重新输入！");
```

由于用到了 JFace 的对话框，在程序的前面要引入相应的包：

```
import org.eclipse.jface.dialogs.*;
```

输入密码时要在密码框显示“*”，需要添加 SWT.PASSWORD 选项：

```
final Text text_1=new Text(shell,SWT.BORDER|SWT.PASSWORD;
```

图 4.44 的中间部分为组件选择面板（Pallete），可以展开相应的文件夹，用拖曳的方法在窗体中加入各种组件。图 4.45 为组件文件夹（SWT Controls）被展开，下面显示的组件被选中后，可加入到窗体中，用拖曳的方法可改变组件的大小和位置。图 4.46 为布局文件夹（SWT Layouts）被展开，可以选择适当的布局方式为窗体布局。

图 4.45 Events 页

图 4.46SWT 组件

图 4.47SWT 布局

⑤运行程序

在 Eclipse 包资源管理器中，右单击文件名，在弹出的菜单中选择【运行方式】→【SWT 应用程序】，运行结果如图 4.48 所示。输入用户名和密码，点击【提交】，则出现用户登录对话框，如图 4.49 所示。如果用户名或密码为空，则出现错误提示对话框，如图 4.50 所示。

图 4.48 用户登录窗体

图 4.49 用户登录对话框

图 4.50 错误信息对话框

⑥完整的源程序

```
package edu.ch4;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import org.eclipse.jface.dialogs.*;
public class SWTDesignerTest {
    public static void main(String[] args) {
        final Display display = Display.getDefault();
        final Shell shell = new Shell();
        shell.setSize(500, 375);
        shell.setText("SWT应用程序");
        shell.open();
        final Text text = new Text(shell, SWT.BORDER);
        text.setBounds(249, 50, 146, 27);
        final Text text_1 = new Text(shell, SWT.BORDER|SWT.PASSWORD);
        text_1.setBounds(249, 147, 146, 27);
        final Label label = new Label(shell, SWT.NONE);
        label.setText("请输入用户名：");
        label.setBounds(71, 50, 118, 27);
        final Label label_1 = new Label(shell, SWT.NONE);
        label_1.setBounds(71, 147, 118, 27);
        label_1.setText("请输入密码：");
        final Button button = new Button(shell, SWT.NONE);
        button.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
                if(text.getText()!=""|text_1.getText()!="")
//调用JFace的对话框显示登录信息
MessageDialog.openInformation(shell,"登录信息","欢迎"+text.getText()+"进入系统！");
                else
//调用JFace的对话框显示出错信息
MessageDialog.openError(shell,"错误","用户名或密码为空，请重新输入！");
            }
        });
        button.setText("提交");
        button.setBounds(74, 245, 115, 27);
        final Button button_1 = new Button(shell, SWT.NONE);
        button_1.setBounds(280, 245, 115, 27);
```

```
        button_1.setText("重置");
        shell.layout();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
    }
}
```

4.7 本章小结

与 AWT 和 SWING 相比，SWT 提供了和本机系统相同的用户界面组件，具有较好的运行效率和稳定的平台表现，因此近年来取得了很快的发展。本章介绍了 SWT 常用组件、SWT 常用容器类、SWT 常用的布局方式、SWT 事件处理方法等图形化用户界面开发技术。最后介绍了一款 SWT 可视化界面开发插件包——SWT Designer。通过大量实例，演示了 SWT 开发桌面应用程序的基本技术和技巧。

4.8 习题

1.试述创建一个典型的 SWT 应用程序常用的步骤。

2.简述 org.eclipse.swt.widgets 包的用途。

3.简述 GridLayout 的常用属性。

4.简述 FormAttachment 类的作用，它有几个构造方法。构造方法中的参数表示什么含义？

5.完成图 4.51 所示图形界面的制作。要求“查询结果”用 group 组件。

图 4.51 数据查询界面

6.完成图 4.52 所示学籍管理主界面的设计与制作。

图 4.52 学籍管理系统主界面

7.在图 4.52 所示界面中，在“用户登录”菜单中添加组件选择事件，当选中“用户登录”时，打开图 4.48 所示用户登录界面。