

SWT 开发参考



作者：Spirit_demon E-mail: itlabers@yahoo.com.cn

文档说明

发布记录

版本	日期	作者	修改内容
1.0	2009-12-3	Spirit_demon	建立初稿

本文档的部分内容来自互联网，一些 SWT 技术专家归纳总结十分到位，在此我只不过将内容重新编排，方便读者阅览，从中受益，文字内容版权归原作者所有。本文档（文本、代码）在免费、且无任何附加条件的前提下，可在网络媒体中自由传播。

如果你对文章中的部分或是全部内容有什么意见或是建议，将您发现的问题和文档改进意见及时反馈给作者。

本文档定位于入门级，同时希望有志之士，能够多提供一些 SWT 方面的更高级的技术资料，能为技术群体无偿贡献自己的所学。

在本文档中 OLE 技术这一章本来是准备写入的，但是笔者在 OLE 技术的理解水平有限，希望有在 Ole 这个技术领域有见解的网友，能和我沟通，完善文档。

在本文档的编写过程中参考如下网站的知识库，希望读者也能通过它们受益

<http://www.ibm.com/developerworks/cn/java/> IBM 中国 java 技术中心

<http://www.java2s.com/> 非常好的一个示例代码参考网站

<http://www.blogjava.net/> 国内非常活跃的 java 社区

目录

1	SWT 简介.....	3
2	SWT 的体系介绍.....	3
3	SWT 包结构.....	4
4	SWT 实例编程.....	6
4.1.	SWT 起步.....	6
4.2.	Button 组件.....	7
4.3.	Label 类组件.....	9
4.4.	Text 组件.....	10
4.5.	Link 组件.....	12
4.6.	Sash 组件.....	13
4.7.	Scale 组件.....	15
4.8.	Slider 组件.....	16
4.9.	ProgressBar 组件.....	18
4.10.	Combo 组件 与 List 组件.....	20
4.11.	Menu 组件.....	22
4.12.	Toolbar 组件.....	25
4.13.	Tray 组件.....	27
4.14.	Tree 组件.....	30
4.15.	Table 组件.....	32
4.16.	TabFolder 组件.....	35
4.17.	MessageBox 组件.....	37
4.18.	ColorDialog 组件.....	39
4.19.	FontDialog 组件.....	41
4.20.	DirectoryDialog 组件.....	43
4.21.	FileDialog 控件.....	44
4.22.	PrintDialog 控件.....	46
4.23.	CCombo 组件.....	48
4.24.	TableEditor 组件.....	49
5	SWT 的组件布局.....	52
5.1.	充满式布局.....	52
5.2.	行列式布局.....	53
5.3.	网格式布局.....	54
5.4.	表格式布局.....	55
6	SWT 事件监听机制.....	56
7	应用 SWT 绘制 2D 图像图像.....	65
8	SWT 的 OpenGL 应用.....	70
9	SWT 和 Swing、AWT 技术比较.....	74
9.1.	AWT 概述.....	74
9.2.	Swing 概述.....	75
9.3.	SWT 概述.....	78
10	结束语.....	84

1 SWT 简介

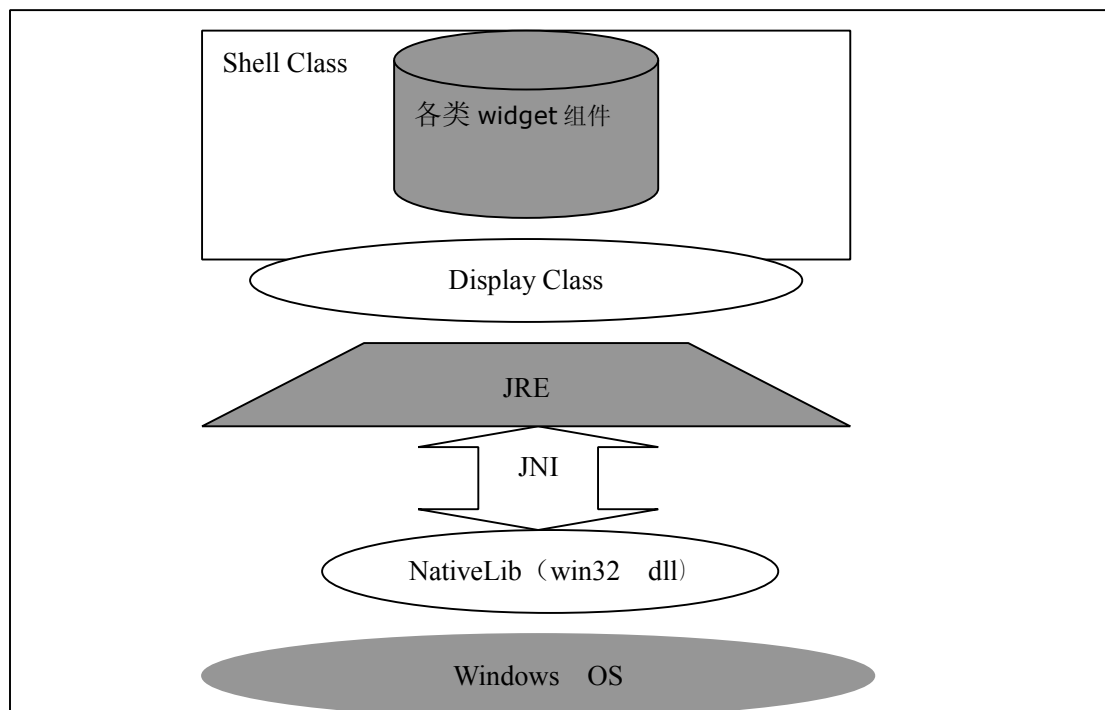
SWT-"Standard Widget Toolkit", 它是一个 Java 平台下开放源码的 Native GUI 组件库, Eclipse 平台 UI 同时也是基于 SWT 和 Jface。SWT 完成了 java GUI 库的轻量级封装,从功能上来说,SWT 与 AWT/SWING 是基本等价的,但是 SWT 以方便有效的方式提供了便携式的(即 Write Once, Run Anywhere)带有本地操作系统观感的 UI 组件,同时借助于 JNI 的调用,有效地改善了 java GUI application 在性能上的不足。

就 java GUI 而言,最早的 AWT 组件现在被认为是样貌丑陋的,而且存在很多问题;SWING 组件虽然也是缺点多多,但是随着 JDK 版本的不断升高,它仍在不断进行着改进,SWT 在 Swing 的技术基础上,借鉴 Swing 的 MVC 优秀的设计理念,以及 AWT 的事件监听机制,解决了由于 widget 系统的固有复杂性以及平台之间微妙的差异,即使在理想情况下,能够达到工业标准的跨平台的 widget 类库也是很难编写和维护的。我认为,SWT 在功能上与 AWT/SWING 不相伯仲,但是组件更为丰富,平台表现稳定,BUG 也相对较少(相对而言)。如果你的应用程序真的需要在多个平台上运行,需要更为美观的界面,又不那么依赖于其他基于 AWT/SWING 的图形库,那么 SWT 或许是一个比 AWT/SWING 更好的选择。

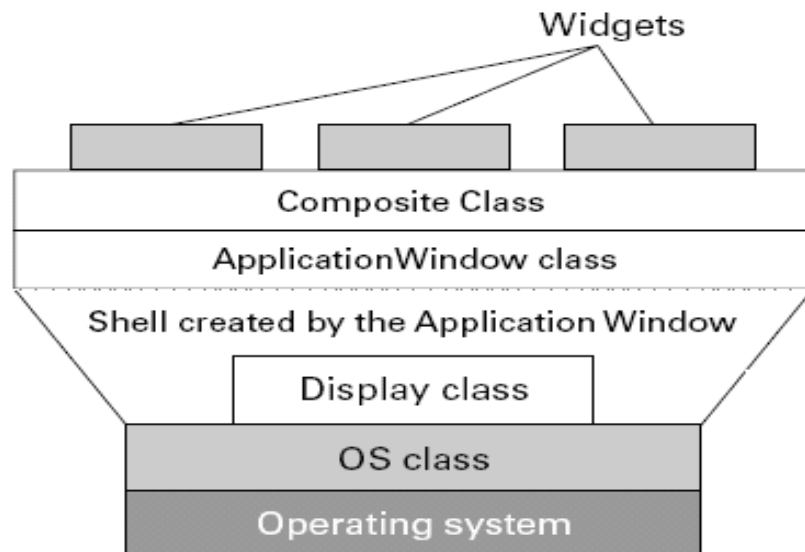
2 SWT 的体系介绍

SWT 由三个基本组件组成:一个本地库,负责与操作系统通讯;一个显示类,作为 SWT 与 GUI 平台通讯的接口;一个 Shell 类,作为程序的顶层窗口,可以容纳窗口小部件(Widget)。下图以 windows 平台为例简述 SWT 的体系结构

SWT 把 Win32 的 API 简单的包装了一下,系统在 SWT 这一层调用的方法、传递的参数被原封不动的代理到了 Win32 层。这是 SWT 的核心思想。



SWT 设计原则那就是 SWT 的 API 一对一的封装 OS 的 API，完全忠实于操作系统的 API 实现的行为，SWT 本身只是调用对应 OS 的 native API，这样在实际运用 SWT 时，避免对相应平台做代码的修改，同时避免潜在的破坏本地化的一些行为。



SWT 拥有标准的与操作系统相同的外观，几乎没有人能看出你的程序是用 Java 写出来的，尽管 AWT 组件也试图组到这一点，但是因为 AWT 组件封装抽象层次原始，在实际的开发中很难达到令人满意的界面效果，SWT 值得推崇的更为重要的一点，其程序运行的效率可以和 VC++ 写出的程序向媲美，而且开发的效率也不在 AWT/SWING 之下。

3 SWT 包结构

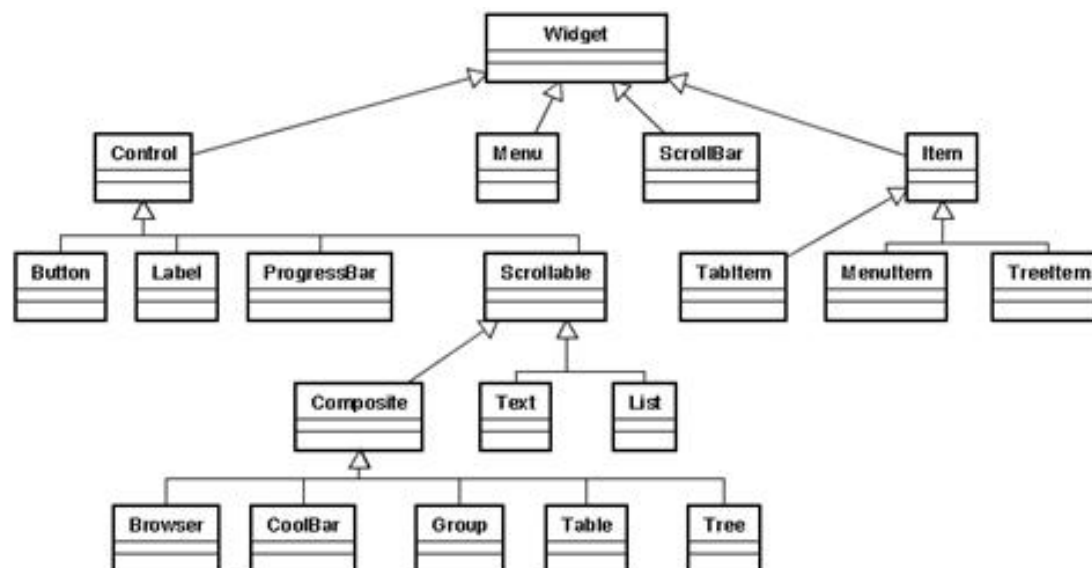
SWT 的所有类都在 org.eclipse.swt 包下。

Core Packages	
org.eclipse.swt	包含 SWT，SWTException 和 SWTError 类。SWT 类中定义了 SWT 中的公共常量，包括部件风格、消息常量等；SWTException 和 SWTError 则定义了 SWT 中异常，错误的基类。
org.eclipse.swt.accessibility	包含一些实现客户端访问控件的消息响应事件接口类。
org.eclipse.swt.custom	包含了一些可自定义的窗口小部件。
org.eclipse.swt.dnd	包含了对拖放操作支持的控制类
org.eclipse.swt.events	包中提供了对 SWT 事件监视器 (Event Listener) 的支持，还有与这些 listener 对应的 Adapter 实现类和 Event 类。

<code>org.eclipse.swt.graphics</code>	包中包含了 SWT 中 2Dgraphic 类，如 Color、Font 和 Image
<code>org.eclipse.swt.internal</code>	包含了对跨平台的原形接口的定义，和描述系统级回调的访问 java 的实例类，跨平台的事件类
<code>org.eclipse.swt.internal.awt.win32</code>	定义了 SWT_AWT 类
<code>org.eclipse.swt.internal.image</code>	图像 PNG 的哈弗曼表，PNG 文件的格式化表达的定义
<code>org.eclipse.swt.internal.ole.win32</code>	跨平台的 SWT 对 OLE 技术的访问封装类
<code>org.eclipse.swt.internal.win32</code>	跨平台数据格式以及，空间信息的表达
<code>org.eclipse.swt.layout</code>	包含了 SWT 的布局管理类
<code>org.eclipse.swt.ole.win32</code>	Win32 平台下 SWT 对 OLE 技术的访问封装类
<code>org.eclipse.swt.printing</code>	提供了对打印的支持
<code>org.eclipse.swt.program</code>	操作系统中相关文件拓展的访问
<code>org.eclipse.swt.widgets</code>	SWT 的小部件，SWT 的核心类

SWT 组件介绍

最重要的类就是 Widget，它是所有界面对象的基类，类图如图所示。



SWT 的 Widget 类结构图

Widget 的直接子类有 Caret(插入光标)、Menu(菜单)、ScrollBar(滚动条)、Tray(系统托盘图标)等。Widget 的子类 Item 下的类是一些无法独立于其他部件的部件，比如 MenuItem(菜单项)、TableItem(表格项)、TrayItem(系统托盘图标项)、TreeItem(树项)等。

Widget 的子类 Control 是一个比较庞大的基类，大部分 SWT 部件都在此类下，其直接子类

有 Button(按钮)、Label(标记)、ProgressBar(进度条)等。Control 的子类 Scrollable 是所有可以带滚动条的对象的基类，比如 Text(文本框)、List(列表框)等。Scrollable 的子类 Composite 是 SWT 中一个重要的类，它是所有可以容纳其他部件的类的基类，其子类有 Browser(浏览器)、Combo(下拉列表框)、Group(组合框)、Table(表格)、Tree(树)等。

4 SWT 实例编程

4.1. SWT 起步

SWT 的 HelloWorld

通常介绍编程实例，我们都需要来一个 HelloWorld 来帮助我们入门，SWT 的 HelloWorld 如下：

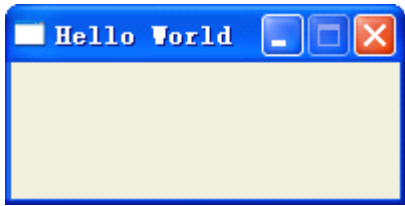
```
package example.e01;

import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;;
public class HelloWorld {

    public static void main(String[] args) {

        Display display=new Display();           // 创建 Display 类对象
        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);              //设置窗体的位置
        shell.setSize(200,100);                   //设置窗体大小
        shell.setText("Hello World");             //设置窗体标题
        shell.open();                             //显示窗体
        while (!shell.isDisposed())              //检验窗体是非关闭
        {
            if (!display.readAndDispatch())       //检验 Display 线程状态是非忙
                display.sleep ();                 //Display 类线程休眠
        }
        display.dispose ();                       //注销 Display 对象资源
    }
}
```

运行这个程序就会得到如下结果：



- 这是一个 SWT 构建 UI 最简单的一个例子，但是却勾勒出 SWT 建立 UI 的一般过程：
- ①创建一个 Display ，相当于建立一个 application 对象实例，与 OS 图形交互数据。
 - ②创建一个或多个 Shell (设置 Shell 的布局) Shell 相当于 windows 的窗口类
 - 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

4.2. Button 组件

按钮 (Button)组件是 SWT 中最常用的组件,按钮组件也是最醒目,直观的组件。按下 Button 完成相应的任务。

Button 组件的构造方法是：

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。

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，在 SWT 类定义了构造样式如：

Button 组件常用式样

样式	说明
SWT.NONE	默认按钮
SWT.PUSH	按钮
SWT.CHECK	多选按钮
SWT.RADIO	单选按钮
SWT.ARROW	箭头按钮
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.TOGGLE	摇摆
SWT.FALUTP	凸起

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)。

Button 组件的常用方法

函数	作用
----	----

setText (String string)	设置按钮文本
setSize (int width,int height)	设置组件的宽度 width，组件高度 height
setLocation (int x,int y)	设置组件的坐标位置(x 轴坐标, y 轴坐标,)
setImage (Image image)	设置按钮以 image 图像填充
setAlignment (int alignment)	设置按钮文本的对齐方式
setBackground (Color color)	设置按钮的背景色
addListener (int Eventtype , Listener lstener)	设置响应监听事件

实例

```

package example.e01;

import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;

public class ShowButton {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

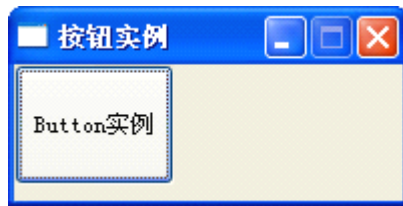
        Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体，风格
        shell.setLocation(200, 200); // 设置窗体的位置
        shell.setSize(200, 100); // 设置窗体大小
        shell.setText("按钮实例"); // 设置窗体标题

        Button button = new Button(shell, SWT.NO); // 构造 Button 对象，设置样式
        button.setLocation(0, 0); // 设置 Button 的位置
        button.setSize(80, 60); // 设置 Button 大小
        button.setText("Button 实例"); // 设置 Button 标题

        shell.open(); // 显示窗体
        while (!shell.isDisposed()) // 检验窗体是非关闭
        {
            if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
                display.sleep(); // Display 类线程休眠
        }
        display.dispose(); // 注销 Display 对象资源
    }
}

```

运行结果如图



4.3. Label 类组件

Label 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，在 UI 中多用为文本提示信息。

Label 组件的构造方法是：

Label (Composite parent,int style) 该方法有两个参数：

第一个参数 parent 是指 Label 创建在哪一个容器上。

第二个参数 style 用来指定 Label 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，

Label 组件的基本样式

样式	说明
SWT.NONE	默认式样
SWT.CENTER	文字居中
SWT.RIGHT	文字靠右
SWT.LEFT	文字靠左
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

Label 组件的常用方法

函数	作用
setText (String string)	设置按钮文本
setSize (int width,int height)	设置组件的宽度 width，组件高度 height
setLocation (int x,int y)	设置组件的坐标位置(x 轴坐标, y 轴坐标,)
setImage (Image image)	设置按钮以 image 图像填充
setAlignment (int alignment)	设置按钮文本的对齐方式
setBackground (Color color)	设置按钮的背景色
addListener (int Eventtype , Listener listener)	设置响应监听事件

实例代码

```
package example.e01;

import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;
```

```

public class ShowLabel {

    public static void main(String[] args) {
        Display display=new Display();           // 创建 Display 类对象

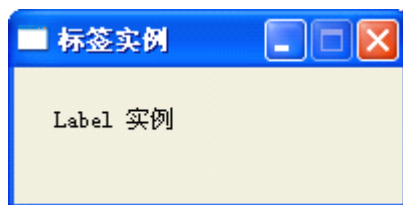
        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);              //设置窗体的位置
        shell.setSize(200,100);                  //设置窗体大小
        shell.setText("标签实例");               //设置窗体标题

        Label label=new Label(shell,SWT.NO);     //构造 Label 对象，设置样式
        label.setLocation(20,20);                 //设置 Label 的位置
        label.setSize(100,50);                   //设置 Label 大小
        label.setText("Label 实例");             //设置 Label 标题

        shell.open();                            //显示窗体
        while (!shell.isDisposed())              //检验窗体是非关闭
        {
            if (!display.readAndDispatch())      //检验 Display 线程状态是非忙
                display.sleep ();               //Display 类线程休眠
        }
        display.dispose ();                      //注销 Display 对象资源
    }
}

```

运行结果如图



4.4. Text 组件

文本组件在图形用户界面应该是运用非常普遍，接受用户输入的数据基本都是通过文本组件实现的，作用相当于 `JTextField` `JTextArea`。同样文本组件 `Text` 的构造方式和 `button`、`Label` 的基本相同。只不过在文本框的样式上会略有区别；

文本框（`Text` 类）的式样如下：

样式	说明
SWT.NONE	默认式样
SWT.CENTER	文字居中
SWT.RIGHT	文字靠右
SWT.LEFT	文字靠左

SWT.MULTI	可以输入多行，须回车换行
SWT.WRAP	自动换行
SWT.BORDER	深陷型
SWT.V_SCROLL	带垂直滚动条。
SWT.H_SCROLL	带水平滚动条

Text 组件的常用方法

函数	作用
setText (String string)	设置按钮文本
setSize (int width,int height)	设置组件的宽度 width，组件高度 height
setLocation (int x,int y)	设置组件的坐标位置(x 轴坐标, y 轴坐标,)
setImage (Image image)	设置按钮以 image 图像填充
setAlignment (int alignment)	设置按钮文本的对齐方式
setBackground (Color color)	设置按钮的背景色
addListener (int Eventtype , Listener lstener)	设置响应监听事件

实例代码

```
package example.e01;

import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import org.eclipse.swt.SWT;;
public class ShowText {

    public static void main(String[] args) {
        Display display=new Display();        // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);            //设置窗体的位置
        shell.setSize(200,100);                //设置窗体大小
        shell.setText("文本框实例");            //设置窗体标题

        Text text=new Text(shell,SWT.NO); //构造 Text 对象，设置样式
        text.setLocation(50,0);                //设置 Text 的位置
        text.setSize(100,50);                  //设置 Text 大小
        text.setText("Text 实例");              //设置 Text 标题

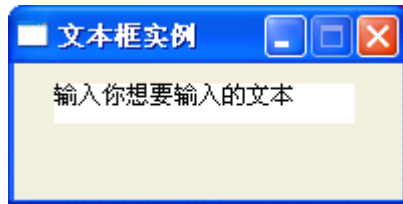
        shell.open();                          //显示窗体
        while (!shell.isDisposed())            //检验窗体是非关闭
        {
            if (!display.readAndDispatch())    //检验 Display 线程状态是非忙
                display.sleep ();              //Display 类线程休眠
        }
    }
}
```

```

        display.dispose ();           //注销 Display 对象资源
    }
}

```

运行结果如图



4.5. Link 组件

Link 组件的作用是实现一个带连接的标签，许多时候他和 label 的作用是一样的，唯一的区别就是在 link 的 text 属性中识别<a> 这一对标签你可以构造像浏览器中的超链接一样的效果。

文本框（Text 类）的式样如下：

样式	说明
SWT.NONE	默认式样

Text 组件的常用方法

函数	作用
setText (String string)	设置按钮文本
setSize (int width,int height)	设置组件的宽度 width，组件高度 height
setLocation (int x,int y)	设置组件的坐标位置(x 轴坐标, y 轴坐标,)
setBackground (Color color)	设置按钮的背景色
setBackgroundImage (Image image)	设置按钮的背景图像
addListener (int Eventtype , Listener listener)	设置响应监听事件

实例

```

package example.e01;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Color;
import org.eclipse.swt.graphics.RGB;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Link;
import org.eclipse.swt.widgets.Shell;

public class ShowLink {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象
    }
}

```

```
Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体, 风格
shell.setLocation(200, 200); // 设置窗体的位置
shell.setSize(200, 100); // 设置窗体大小
shell.setText("link 实例"); // 设置窗体标题

Link link = new Link(shell, SWT.Activate); // 构造Link对象, 设置
样式

link.setText("<a>Link实例</a>"); // 设置Link标题
link.setForeground(new Color(null, new RGB(255, 0, 0))); //设置前置
色

link.setBounds(25, 18, 131, 36); //设置坐标,宽度、高度
(x,y,width, height)
link.setFont(new Font(null, "仿宋_GB2312", 24, 10)); //设置字体
shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}
display.dispose(); // 注销 Display 对象资源

}
```

运行结果如图



4.6. Sash 组件

Sash 组件在图形用户界面反映的是背景网格，在文本编程之中运用的较多，将文本之间的层次感显示出来。**Sash** 能够在其 **Parent** 容器里进行拖拽并产生条纹线的显示效果，在容器间隙中，能直观的容器大小的平滑过渡变化。

带状框（Sash 类）的式样如下：

样式	说明
SWT.NONE	默认式样
SWT. HORIZONTAL,,	水平
SWT. VERTICAL	竖直
SWT. SMOOTH	流动

Sash 的常用方法:

函数	作用
setSize (int width,int height)	设置组件的宽度 width，组件高度 height
setLocation (int x,int y)	设置组件的坐标位置(x 轴坐标, y 轴坐标,)
setBackground (Color color)	设置按钮的背景色
addListener (int Eventtype , Listener lstener)	设置响应监听事件

实例代码

```
package example.e01;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Color;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Sash;
import org.eclipse.swt.widgets.Shell;
public class ShowSash {

    public static void main(String[] args) {
        Display display=new Display();        // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);            //设置窗体的位置
        shell.setSize(200,100);                //设置窗体大小
        shell.setText("sash 实例");            //设置窗体标题

        Sash sash=new Sash(shell,SWT.NO); //构造 Sash 对象，设置样式
        sash.setLocation(20,20);                //设置 Sash 的位置
        sash.setSize(100,20);                  //设置 Sash 大小
        sash.setVisible(true);                  //设置可见度
        sash.setBackground(new Color(null,255,0,255)); //设置网格的背景颜色
        shell.open();                            //显示窗体
        while (!shell.isDisposed())            //检验窗体是非关闭
        {
            if (!display.readAndDispatch())    //检验 Display 线程状态是非忙
                display.sleep ();              //Display 类线程休眠
        }
        display.dispose ();                    //注销 Display 对象资源
    }
}
```

运行结果如图



4.7. Scale 组件

Scale 组件在图形用户界面反映的是数值的变化，在选择连续变化的数值运用的较多。
滑动杆（Scale 类）的式样如下：

样式	说明
SWT.NONE	默认式样 水平滑动
SWT.HORIZONTAL	水平滑动
SWT.VERTICAL	竖直滑动

Scale 组件的常用方法:

函数	作用
setMaximum (int value)	设置滑动框的值的最大上限
setMinimum (int value)	设置滑动框的值的最小下限
setIncrement (int value)	设置 value 作为步进
setSelection (int value)	设置组件滑动的 value 值的位置，按照百分比来定位
setSize (int width,int height)	设置组件的宽度 width，组件高度 height
setLocation (int x,int y,)	设置组件的坐标位置(x 轴坐标, y 轴坐标,)
addListener (int Eventtype , Listener lstener)	设置响应监听事件

实例代码

```
package example.e01;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Scale;
import org.eclipse.swt.widgets.Shell;
public class ShowScale {

    public static void main(String[] args) {
        Display display=new Display();        // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);            //设置窗体的位置
        shell.setSize(200,200);                //设置窗体大小
        shell.setText("滑动条");                //设置窗体标题
    }
}
```

直

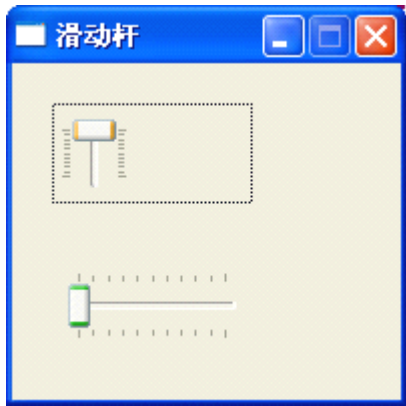
```
Scale v_scale=new Scale(shell,SWT.VERTICAL ); //构造 Scale 对象， 设置样式 竖
v_scale.setLocation(20,20); //设置 Scale 的位置
v_scale.setSize(100,50); //设置 Scale 大小
v_scale.setVisible(true);
```

水平

```
Scale h_scale=new Scale(shell,SWT.HORIZONTAL); //构造 Scale 对象， 设置样式
h_scale.setLocation(20,100); //设置 Scale 的位置
h_scale.setSize(100,50); //设置 Scale 大小
h_scale.setVisible(true);
```

```
shell.open(); //显示窗体
while (!shell.isDisposed()) //检验窗体是非关闭
{
    if (!display.readAndDispatch()) //检验 Display 线程状态是非忙
        display.sleep (); //Display 类线程休眠
}
display.dispose (); //注销 Display 对象资源
}
```

运行效果如图



4.8. Slider 组件

Slider 组件在图形用户界面反映的是数值的变化，在选择连续变化的数值运用的较多。
滑动边框 （Slider 类）的式样如下：

样式	说明
SWT.NONE	默认式样 水平滑动
SWT. HORIZONTAL	水平滑动
SWT. VERTICAL	竖直滑动

Slider 组件的常用方法

函数	作用
setMaximum (int value)	设置滑动框的值的最大上限
setMinimum (int value)	设置滑动框的的最小下限
setIncrement (int value)	设置 value 作为步进
setSelection (int value)	设置组件滑动的 value 值的位置, 按照百分比来定位
setSize (int width,int height)	设置组件的宽度 width, 组件高度 height
setLocation (int x,int y,)	设置组件的坐标位置(x 轴坐标, y 轴坐标,)
setPageIncrement (int value)	设置每一刻度的步进量
addListener (int Eventtype , Listener listener)	设置响应监听事件

实例代码

```
package example.e01;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Slider;
public class ShowSlider {

    public static void main(String[] args) {
        Display display=new Display();        // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体, 风格
        shell.setLocation(200, 200);            //设置窗体的位置
        shell.setSize(200,200);                //设置窗体大小
        shell.setText("滑动框实例");           //设置窗体标题

        Slider v_slider=new Slider(shell,SWT.VERTICAL ); //构造 Slider 对象, 设置样式 竖
直
        v_slider.setLocation(20,20);            //设置 Slider 的位置
        v_slider.setSize(20,100);               //设置 Slider 大小
        v_slider.setPageIncrement(10);          //设置每一个刻度的增进值
        v_slider.setMaximum(100); //设置滑动值的最高上限
        v_slider.setMinimum(0);//设置滑动值的最小下限

        Slider h_slider=new Slider(shell,SWT.HORIZONTAL); //构造 Slider 对象, 设置样式
水平
        h_slider.setLocation(50,100);           //设置 Slider 的位置
        h_slider.setSize(100,20);              //设置 Slider 大小
        h_slider.setPageIncrement(10);          //设置每一个刻度的增进值
        h_slider.setMaximum(100); //设置滑动值的最高上限
    }
}
```

```
h_slider.setMinimum(0);//设置滑动值的最小下限

shell.open();                //显示窗体
while (!shell.isDisposed())  //检验窗体是非关闭
{
    if (!display.readAndDispatch()) //检验 Display 线程状态是非忙
        display.sleep ();          //Display 类线程休眠
}
display.dispose ();          //注销 Display 对象资源
}
```

运行效果如图



4.9. ProgressBar 组件

进度条（ProgressBar）是一个显示任务进度的控件，用户可以通过观察进度刻度的滑动，明确的知道任务进展状况，严格来说这个组件并没有实际功能意义，只是在用户观感上、心理上能起到非常好的效果。

进度条（ProgressBar 类）的式样如下：

样式	说明
SWT.NONE	默认式样 水平滑动
SWT. HORIZONTAL	水平滑动
SWT. VERTICAL	竖直滑动

ProgressBar 组件的常用方法

函数	作用
setMaximum (int value)	设置滑动框的值的最大上限
setMinimum (int value)	设置滑动框的值的最小下限
setIncrement (int value)	设置 value 作为步进
setSelection (int value)	设置组件滑动的 value 值的位置，按照百分比来定位
setSize (int width,int height)	设置组件的宽度 width，组件高度 height

setLocation (int x,int y)	设置组件的坐标位置(x 轴坐标, y 轴坐标,)
addListener (int EventType , Listener listener)	设置响应监听事件

实例代码

```

package example.e01;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.ProgressBar;
import org.eclipse.swt.widgets.Shell;

public class ShowProgressBar {

    public static void main(String[] args) {
        final Display display=new Display();        // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);        //建立 shell 主窗体, 风格
        shell.setLocation(200, 200);        //设置窗体的位置
        shell.setSize(200,200);        //设置窗体大小
        shell.setText("进度条实例");        //设置窗体标题

        final ProgressBar bar = new ProgressBar(shell, SWT.NONE); //新建进度条对象
        bar.setSize(200, 32);        //设置进度条大小
        bar.setMaximum(100);        //设置进度条最大上限
        bar.setMinimum(0);        //设置最小下限
        final int maximum = bar.getMaximum();
        new Thread() {                //新建一个子线程改变 ProgressBar 控件的
select 值
            public void run() {

                for (int i = 0; i <= maximum; i++) {
                    try {
                        Thread.sleep(100);        //线程休眠 100 毫秒

                        final int index = i;

                        display.asyncExec(new Runnable() {
                            //Display 线程中更新 UI 上 ProgressBar 的改变
                            public void run() {
                                bar.setSelection(index);
                            }
                        });
                    } catch (Throwable th) {}
                }
            }
        }
    }
}

```

```
        }
    }
    }.start();
    shell.pack();
    shell.open();
    while (!shell.isDisposed()) // 检验窗体是非关闭
    {
        if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
            display.sleep(); // Display 类线程休眠
    }
    display.dispose(); // 注销 Display 对象资源
}
}
```

运行结果如图



4.10. Combo 组件 与 List 组件

下拉框组件，和列表框组件在访问数据的方法上有很多的相识之处，所以将两种组件放在一起介绍。

下拉框（Combo 类）的式样

样式	说明
SWT.NONE	默认式样
SWT.READ_ONLY	只读
SWT.SIMPLE	带水平滚动条

列表框（List 类）的式样

样式	说明
SWT.NONE	默认式样
SWT.V_SCROLL	带水平滚动条
SWT.MULTI	允许复选
SWT.SINGLE	允许单选

下拉框（Combo 类）的常用方法

函数	作用
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
addListener (int Eventtype , Listener lster)	设置响应监听事件

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。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。

实例代码

```
package example.e01;

import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.List;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;

public class ShowComboandList {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

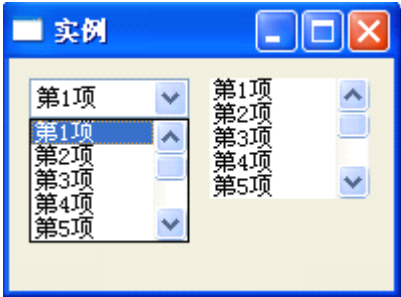
        Shell shell = new Shell(display); //建立 shell 主窗体，风格
        shell.setLocation(200, 200); //设置窗体的位置
        shell.setSize(200, 150); //设置窗体大小
        shell.setText("实例"); //设置窗体标题

        Combo combo = new Combo(shell, SWT.MULTI |
        SWT.V_SCROLL|SWT.READ_ONLY); //设置 Combo 下拉框对象
        combo.setLocation(10, 10); //设置下拉框的位置
        combo.setSize(80, 60); //设置组件大小
        combo.removeAll(); //清空数据
        for (int i = 0; i < 10; i++) {
            combo.add("第" + (i + 1) + "项"); //新增数据项
        }
        combo.select(0);

        List list = new List(shell, SWT.MULTI | SWT.V_SCROLL); //设置 List 列表框对象
        list.setLocation(100, 10); //设置列表框的位置
        list.setSize(80, 60); //设置组件大小
        list.removeAll(); //清空数据
        for (int i = 0; i < 10; i++) {
            list.add("第" + (i + 1) + "项");
        }
    }
}
```

```
shell.open(); //显示窗体
while (!shell.isDisposed()) //检验窗体是非关闭
{
    if (!display.readAndDispatch()) //检验 Display 线程状态是非忙
        display.sleep(); //Display 类线程休眠
}
display.dispose(); //注销 Display 对象资源
}
```

运行效果



4.11. Menu 组件

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）。应用程序中的菜单作用就是为了实现模块功能的导航，相比 Tree 组件，Menu 导航的功能域更大一些。能清晰地反映响应的功能所在的位置，像 Word 这样的软件，对于菜单的管理是很规范的，每一菜单下面的功能都是划分的很清晰。

Menu 的样式如下

样式	说明
SWT.NONE	默认式样
SWT.BAR	菜单栏
SWT.DROP_DOWN	下拉弹出 同默认样式
SWT.POP_UP	弹出
SWT.NO_RADIO_GROUP	非单选
SWT.LEFT_TO_RIGHT	自左向右
SWT.RIGHT_TO_LEFT	自右向左

MenuItem 样式如下

样式	说明
SWT.NONE	默认式样
SWT.CASCADE	级联
SWT.CHECK	多选

SWT. PUSH	默认弹出
SWT. RADIO	单选
SEPARATOR	分隔符

Menu 常用函数

函数	作用
getItem (int index)	获取 index 对应的 MenuItem
getItemCount ()	获取 MenuItem 的数量
getItems ()	获取 Menu 容器 menuItem 的对象组
getParent ()	获取上级组件
getShell ()	获取上级 Shell
setVisible ()/ setVisible (boolean visible)	获取、设置可见性
setLocation (int x, int y)	设置组件的坐标
setEnabled (boolean enabled)	设置组件的可用性

实例代码

```
package example.e02;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.widgets.MenuItem;
import org.eclipse.swt.widgets.Shell;
public class ShowMenu {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

        Shell shell = new Shell(display); // 建立 shell 主窗体，风格样式
        shell.setLocation(200, 200); // 设置窗体的位置
        shell.setSize(400, 300); // 设置窗体大小
        shell.setText("菜单实例"); // 设置窗体标题

        Menu mainmenu = new Menu(shell, SWT.BAR); // 构造主菜单容器对象，设置样式"
工具"
        shell.setMenuBar(mainmenu);

        //构建第一个子菜单节点
        MenuItem menuItem1 = new MenuItem(mainmenu, SWT.CASCADE); // 构造主菜单
的子菜单节点对象，设置样式"级联"
        menuItem1.setText("菜单 1"); //设置菜单显示文本
        Menu menu1 = new Menu(shell, SWT.DROP_DOWN); // 构造子菜单容器对象，设置
样式
        menuItem1.setMenu(menu1); //将菜单容器关联到子菜单对象上
```

```

MenuItem treeitem3 = new MenuItem(menu1, SWT.NO); // 构造子菜单节点对象，设置样式默认
treeitem3.setText("子菜单一");//设置菜单显示文本
MenuItem treeitem4 = new MenuItem(menu1, SWT.NO); // 构造子菜单节点对象，设置样式默认
treeitem4.setText("子菜单二");//设置菜单显示文本

//构建第二个子菜单节点
MenuItem menuitem2 = new MenuItem(mainmenu, SWT.CASCADE); // 构造子菜单节点对象，设置样式默认
menuitem2.setText("菜单 2");

Menu menu2 = new Menu(shell, SWT.DROP_DOWN); // 构造子菜单容器对象，设置样式
menuitem2.setMenu(menu2);

MenuItem treeitem5 = new MenuItem(menu2, SWT.CHECK); // 构造子菜单节点对象，设置样式默认
treeitem5.setText("多选子菜单一");
MenuItem treeitem6 = new MenuItem(menu2, SWT.CHECK); // 构造子菜单节点对象，设置样式默认
treeitem6.setText("多选子菜单二");

//构建第三个子菜单节点
MenuItem menuitem3 = new MenuItem(mainmenu, SWT.CASCADE); // 构造子菜单节点对象，设置样式默认
menuitem3.setText("菜单 3");

Menu menu3 = new Menu(shell, SWT.DROP_DOWN); //构造子菜单容器对象，设置样式
menuitem3.setMenu(menu3);

MenuItem treeitem7 = new MenuItem(menu3, SWT.RADIO); // 构造子菜单节点对象，设置样式默认
treeitem7.setText("单选子菜单一");
MenuItem treeitem8 = new MenuItem(menu3, SWT.RADIO); // 构造子菜单节点对象，设置样式默认
treeitem8.setText("单选子菜单二");

shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}

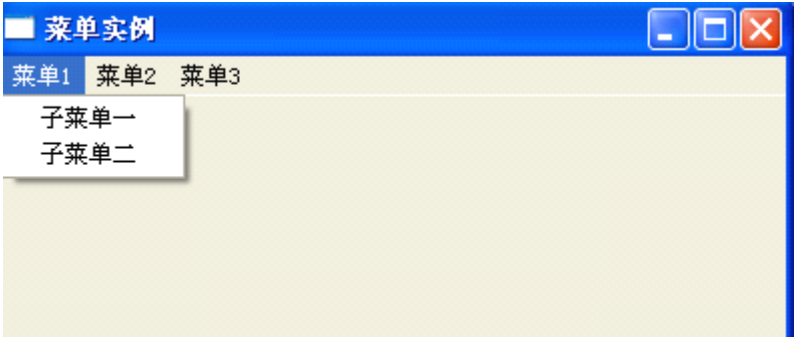
```



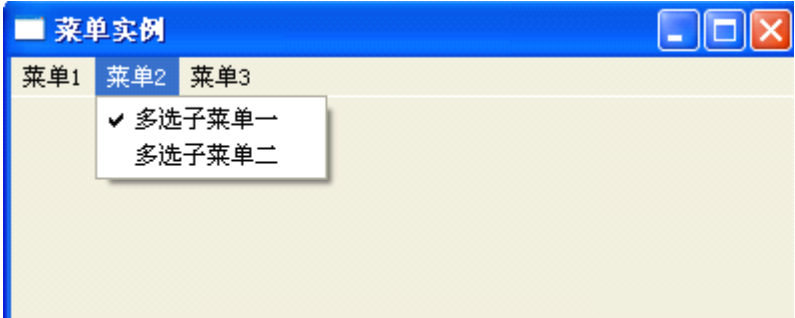
```
}
    display.dispose(); // 注销 Display 对象资源
}
}
```

运行效果

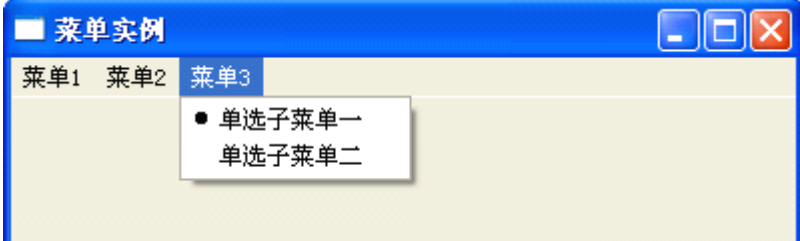
默认样式



多选样式



单选样式



4.12. Toolbar 组件

工具栏（Toolbar 类，ToolItem 类）是常用的 SWT 组件，Toolbar 是一个工具菜单栏，同时也是一个容器，可以容纳工具子项（ToolItem）。应用程序中的工具作用就是为了快速定位功能键，在工具栏中，所有功能几乎都能在对应的 Menu 下中找到，工具栏只是提供了基本常用功能的汇集，能清晰地反映响应的功能，像 Word 这样的软件，对于工具的管理是很规范的，每一工具栏的工具按钮的功能也是很清晰。

Menu 的样式如下

样式	说明
----	----

SWT.NONE	默认式样
SWT.FLAT	工具项是可以流动
SWT.WRAP	工具项可互换位置
SWT.RIGHT	右对齐
SWT.HORIZONTAL	水平样式
SWT.VERTICAL	竖直样式
SWT.SHADOW_OUT	带突出阴影

ToolBar 常用函数

样式	说明
SWT.NONE	默认式样
SWT.CASCADE	级联
SWT.CHECK	多选
SWT.PUSH	默认弹出
SWT.RADIO	单选
SEPARATOR	分隔符

ToolBar 常用函数

函数	作用
getItem (int index)	获取 index 对应的 ToolItem
getItemCount ()	获取 ToolItem 的数量
getItems ()	获取 Toolbar 容器 ToolItem 的对象组
getParent ()	获取上级组件
getShell ()	获取上级 Shell
setVisible ()/ setVisible (boolean visible)	获取、设置可见性
setLocation (int x, int y)	设置组件的坐标
setEnabled (boolean enabled)	设置组件的可用性

实例代码

```
package example.e02;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.ToolBar;
import org.eclipse.swt.widgets.ToolItem;
public class ShowToolBar {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

        Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体，风格样式
```

```

shell.setLocation(200, 200); // 设置窗体的位置
shell.setSize(200, 100); // 设置窗体大小
shell.setText("工具栏实例"); // 设置窗体标题

//构建第一个子菜单节点
ToolBar toolbar = new ToolBar(shell, SWT.WRAP); // 构造工具栏对象，设置样式"级
联"
toolbar.setSize(200,200);

Button button1=new Button(toolbar,SWT.NO);
button1.setText("工具按钮一");

Button button2=new Button(toolbar,SWT.NO);
button2.setText("工具按钮二");

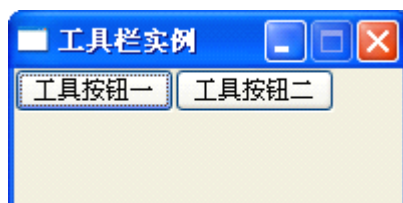
ToolItem toolitem1 = new ToolItem(toolbar, SWT.SEPARATOR); // 构造工具项对象，
设置样式
toolitem1.setControl(button1);//将工具项设置为 Button
toolitem1.setWidth(80);

ToolItem toolitem2 = new ToolItem(toolbar, SWT.SEPARATOR); // 构造工具项对象，
设置样式
toolitem2.setControl(button2);//将工具项设置为 Button
toolitem2.setWidth(80);

shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}
display.dispose(); // 注销 Display 对象资源
}
}

```

运行效果



4.13. Tray 组件

系统托盘就是计算机右下角的那部分（任务栏最右面）。系统托盘（`tray` 类，`trayItem` 类）是常用的 SWT 组件，`trayitem` 是托盘容器，可以容纳工具子项（`Menu`）。

系统托盘驻留程序一般都是在计算机后台运行，像 QQ、msn 等等程序，都会出现在系统托盘中。一般程序只在系统托盘中运行，那么比较节省资源，但是如果是不会用到的程序，最好还是将其关闭退出。

Tray 的样式、TrayItem 样式没有特别定义

样式	说明
SWT.NONE	默认式样

对于 Tray 的操作极少，在托盘中增加菜单，是对相应的 TrayItem 操作，和 `menu`、`MenuItem` 的关系有点相像

Tray 常用函数

函数	作用
getItem (int index)	获取 index 对应的 trayitem
getItemCount ()	获取托盘 trayitem 的数量
getItems ()	获取 trayitem 容器子对象组

TrayItem 常用函数

函数	作用
getParent ()	获取上级的 tray
getToolTip ()	获取托盘对用的 ToolTip
getToolTipText () / getToolTipText (String tip)	获取、设置 ToolTip 的提示文本
setVisible () / setVisible (boolean visible)	获取、设置组件的可见性
setImage (Image image)	设置 TrayItem 的 Image
getText () / setText (String text)	获取设置 Tray 的 title
getMenu ()	获取 trayitem 容器 Menu 的对象组

实例代码

```
package example.e02;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.MenuDetectEvent;
import org.eclipse.swt.events.MenuDetectListener;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.events.SelectionListener;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.widgets.MenuItem;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Tray;
import org.eclipse.swt.widgets.TrayItem;
```

```
public class ShowTray {

    public static void main(String[] args) {
        final Display display = new Display(); // 创建 Display 类对象

        final Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体，风格样式
        shell.setLocation(200, 200); // 设置窗体的位置
        shell.setSize(400, 300); // 设置窗体大小
        shell.setText("托盘实例"); // 设置窗体标题

        final Menu menu = new Menu(shell);
        final MenuItem mi1 = new MenuItem(menu, SWT.PUSH);
        mi1.setText("隐藏窗口");
        mi1.addSelectionListener(new SelectionListener() {
            public void widgetDefaultSelected(SelectionEvent e) {
                widgetSelected(e);
            }
            public void widgetSelected(SelectionEvent e) {
                if (shell.isVisible()) {
                    shell.setVisible(false);
                    mi1.setText("显示窗口");
                } else {
                    shell.setVisible(true);
                    mi1.setText("隐藏窗口");
                    shell.forceActive();
                }
            }
        });

        MenuItem quit = new MenuItem(menu, SWT.PUSH);
        quit.setText("退出");
        quit.addListener(SWT.Selection, new Listener() {
            public void handleEvent(Event e) {
                display.dispose();
                System.exit(0);
            }
        });

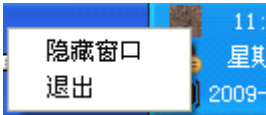
        Tray tray = display.getSystemTray(); // 构造 Tree 对象，设置样式

        TrayItem trayitem1 = new TrayItem(tray, SWT.DROP_DOWN); // 构造 TreeItem 对象，设置样式
        trayitem1.setText("托盘");
        trayitem1.setToolTipText("系统托盘");
    }
}
```

```
trayitem1.setImage(new Image(display, "C://WINDOWS//Coffee Bean.bmp"));

trayitem1.addMenuDetectListener(new MenuDetectListener() {
    public void menuDetected(MenuDetectEvent arg0) {
        menu.setLocation(display.getCursorLocation());
        menu.setVisible(true);
    }
});
shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}
display.dispose(); // 注销 Display 对象资源
}
```

运行结果如图



4.14. Tree 组件

Tree 组件在 UI 这一块的应用，多半集中在界面的导航，Tree 的结构简单，有效的表达项目与项目之间的前驱和后继关系，以及项目与子项目之间的包含关系，所以，在 UI 的开发中，Tree 组件是最理性用于导航的。

对于 Tree 这种组件，其核心在于他作为一个容器了来承载叶子节点，SWT 将应用容器封装在 Tree 这个类中，叶子节点的应用封装在 TreeItem 这个类中，这样命名很像 Borland 的一些类封装的命名法则，在 borland 的 VCL 中，有类似的命名规范，至于两者到底是借鉴哪一种共同的命名规则，可以查一下，在下面即将介绍的 Menu 中，同样有 MenuItem 的命名传统。

对于 Tree 组件而言，不做过多地对于样式的说明

样式	说明
SWT.NONE	默认式样
SWT.SINGE	允许单选
SWT. MUTI	允许复选
SWT. CHECK	允许多选

对于 Ttree 组件的访问就是就是对于根节点的访问。

在 Tree 中提供了常用方法

函数	作用
getItem(int index)	获取 index 对应的 trayitem

getItemCount()	获取托盘 trayitem 的数量
getItems()	获取 trayitem 容器子对象组
getBackground()	提供访问 Tree 的背景，对应的有 setBackground(Color color)
getParent()	获取根节点
setImage(Image image)	设置根节点的 Image 图像

在 TreeItem 中提供了一些方法

函数	作用
getBackground()	提供访问 TreeItem 的背景，对应的有 setBackground(Color color)
getItemCount()	获取叶子节点 TreeItem 的数目
getItem()	获取叶子节点 TreeItem 对象组成的数组
getParent()	获取根节点 Tree
setImage(Image image)	设置根节点的 Image 图像
setText(String title)	设置当前节点的显示文本

实例代码

```
package example.e02;

import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Tree;
import org.eclipse.swt.widgets.TreeItem;
import org.eclipse.swt.SWT;

public class ShowTree {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

        Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体，风格
        shell.setLocation(200, 200); // 设置窗体的位置
        shell.setSize(400, 300); // 设置窗体大小
        shell.setText("Tree 实例"); // 设置窗体标题

        Tree tree = new Tree(shell, SWT.SINGLE); // 构造 Tree 对象，设置样式
        tree.setLocation(0, 0); // 设置 Tree 对象的位置
        tree.setSize(200, 200); // 设置 Tree 对象大小

        TreeItem treeitem1 = new TreeItem(tree, SWT.NO); // 构造 TreeItem 对象，设置样式
        treeitem1.setText("树的一棵子树"); // 设置子节点显示的文本
        TreeItem treeitem2 = new TreeItem(tree, SWT.NO); // 构造 TreeItem 对象，设置样式
        treeitem2.setText("树的二棵子树"); // 设置子节点显示的文本
    }
}
```

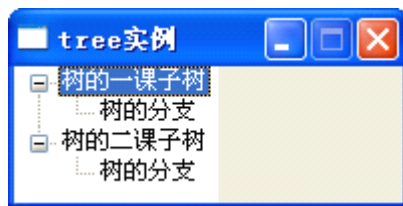
```

        TreeItem treeitem3 = new TreeItem(treeitem1, SWT.NO); // 构造 TreeItem 对象的
TreeItem 对象，设置样式
        treeitem3.setText("第一棵子树的分支");
        TreeItem treeitem4 = new TreeItem(treeitem2, SWT.NO); // 构造 TreeItem 对象的
TreeItem 对象，设置样式
        treeitem4.setText("第一棵子树的分支");

        shell.open(); // 显示窗体
        while (!shell.isDisposed()) // 检验窗体是非关闭
        {
            if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
                display.sleep(); // Display 类线程休眠
        }
        display.dispose(); // 注销 Display 对象资源
    }
}

```

运行结果如图



4.15. Table 组件

Table 控件在 UI 应用中也是一个重要的控件，作为格式化显示数据集合的控件，Table 控件使用是很频繁的，Table 控件能够清晰的显示二维数据，和数据库的数据二维结构对应，所以 Table 中的数据集合通常是和数据库的字段结合使用，能起到很好的效果。

Table 控件二维性通常要结合 TableColumn （数据列） 和 TableItem （数据行）

Table 的样式

样式	说明
SWT.NONE	默认式样
SWT.SINGE	允许单行显示
SWT. MUTI	允许多行显示
SWT. CHECK	允许带多选框
SWT. FULL_SELECTION	允许选择整行多列模式
SWT. HIDE_SELECTION	允许选择整行单列模式

Table 常用方法

函数	作用
getItem (int index)	获取 index 对应的 trayitem
getColumns ()	获取所有列对象
getColumn (int index)	获取 index 对应的单列对象
getColumnCount ()	获取总列数
getItems ()	获取行数对象
getItemCount ()	获取总行数
remove (int index)	移除 index 的列对象

TableColumn 样式

样式	说明
SWT.NONE	默认式样
SWT.LEFT	允许左对齐
SWT.RIGHT	允许右对齐
SWT.CENTER	允许局中对齐

TableItem 样式

样式	说明
SWT.NONE	默认式样

TableItem 方法

函数	作用
setText (String[] data)	设置数据行的值，数组长度为列数
setBackground (Color color)	设置数据行的背景色

实例代码

```
package example.e02;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Table;
import org.eclipse.swt.widgets.TableColumn;
import org.eclipse.swt.widgets.TableItem;
public class ShowTable {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

        Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体，风格
```

```

shell.setLocation(200, 200); // 设置窗体的位置
shell.setSize(250, 100); // 设置窗体大小
shell.setText("Table 实例"); // 设置窗体标题

Table table = new Table(shell, SWT.BORDER | SWT.SINGLE |
SWT.FULL_SELECTION); // 构造表格对象，设置样式
table.setLocation(0, 0); // 设置表格的位置
table.setHeaderVisible(true); //设置列标题
table.setSize(200,200); //设置表格大小

TableColumn tablecolumn1 = new TableColumn(table, SWT.LEFT); // 构造表单元列对
象，设置样式左对齐
tablecolumn1.setText("第一列"); //设置列标题
tablecolumn1.setWidth(100); //设置宽度 默认宽度为 0（通常为不可见）

TableColumn tablecolumn2 = new TableColumn(table, SWT.RIGHT); // 构造表单元列对
象，设置样式 右对齐
tablecolumn2.setText("第二列");
tablecolumn2.setWidth(100);

TableItem tableitem1=new TableItem(table,SWT.NONE); //构造数据值对象(行对
象)
tableitem1.setText(new String[]{"ABCDEF","12321312"}); //填充数据项的值

TableItem tableitem2=new TableItem(table,SWT.NONE);
tableitem2.setText(new String[]{"HNFNSF","23749324"});

shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}
display.dispose(); // 注销 Display 对象资源
}
}

```

运行结果如图：



4.16. TabFolder 组件

TableFolder 选项卡组件，在能在同一 Shell 区域显示多个重复的组件容器，各个 Composite 容器位置重叠，特别像浏览器这样的软件，就广泛的利用选项卡，避免重新建立窗口。

TabFolder 的样式

样式	说明
SWT.NONE	默认样式
SWT.TOP,	选项卡在顶部，同默认样式
SWT.BOTTOM	选项卡在底部

TabFolder 方法

函数	作用
getItem (int index)	获取 index 对应的 Tabitem
getItemCount ()	获取托盘 Tabitem 的数量
getItems ()	获取 Tabitem 子对象组
getSelection ()	作用同 getItems ()

TabItem 样式

样式	说明
SWT.NONE	默认样式

TabItem 方法

函数	作用
setImage (Image image)	设置 TabItem 的 Image
getText () / setText (String text)	获取设置 TabItem 的 title
setBackground (Color color)	设置背景色

实例代码

```
package example.e02;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Table;
import org.eclipse.swt.widgets.TableColumn;
import org.eclipse.swt.widgets.TableItem;
public class ShowTable {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象
```

```

Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体，风格
shell.setLocation(200, 200); // 设置窗体的位置
shell.setSize(400, 300); // 设置窗体大小
shell.setText("按钮实例"); // 设置窗体标题

Table table = new Table(shell, SWT.BORDER | SWT.SINGLE |
SWT.FULL_SELECTION); // 构造表格对象，设置样式
table.setLocation(0, 0); // 设置表格的位置
table.setHeaderVisible(true); // 设置列标题
table.setSize(200, 200); // 设置表格大小

TableColumn tablecolumn1 = new TableColumn(table, SWT.LEFT); // 构造表单元对象，设置样式左对齐
tablecolumn1.setText("第一列"); // 设置列标题
tablecolumn1.setWidth(100); // 设置宽度 默认宽度为 0（通常为不可见）

TableColumn tablecolumn2 = new TableColumn(table, SWT.RIGHT); // 构造表单元对象，设置样式 右对齐
tablecolumn2.setText("第二列");
tablecolumn2.setWidth(100);

TableItem tableitem1 = new TableItem(table, SWT.NONE); // 构造数据值对象(行对象)
tableitem1.setText(new String[]{"ABCDEF", "12321312"}); // 填充数据项的值

TableItem tableitem2 = new TableItem(table, SWT.NONE);
tableitem2.setText(new String[]{"HNFNSF", "23749324"});

shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是否关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是否忙
        display.sleep(); // Display 类线程休眠
}
display.dispose(); // 注销 Display 对象资源
}

```

运行结果如图



4.17. MessageBox 组件

消息提示对话框，用于消息提示，最简单的对话框，直观的显示文本。

MessageBox 的样式

样式	说明
SWT.NONE	默认式样
SWT.ICON_ERROR	带错误图标
SWT.ICON_INFORMATION	信息提示
SWT.ICON_QUESTION	带问号图标
SWT.ICON_WARNING,	带警告图标
SWT.ICON_WORKING	带忙状态图标
SWT.OK	带“是”按钮
SWT.OK CANCEL	带“是”、“否”按钮
SWT.YES NO	带“是”、“否”按钮
SWT.YES NO CANCEL	带“是”、“否”、“取消”按钮
SWT.RETRY CANCEL	带“重试”、“取消”按钮
SWT.ABORT RETRY IGNORE	带“放弃”、“重试”、“忽略”按钮

- Messagebox 的常用方法
- getMessage() 获取对话框中的消息
 - open() 获取打开对话框
 - setMessage(String message) 设置对话框中的消息
 - setText(String title) 设置消息对话框中的标题
 - getText() 获取消息对话框中的标题

```
package example.e03;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.MessageBox;
import org.eclipse.swt.widgets.Shell;

public class ShowMessageBox {
```

```

public static void main(String[] args) {
    Display display=new Display();           // 创建 Display 类对象

    Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
    shell.setLocation(200, 200);              //设置窗体的位置
    shell.setSize(200,100);                  //设置窗体大小
    shell.setText("消息对话框");             //设置窗体标题

    MessageBox messagebox1=new MessageBox(shell,SWT.NONE); //构造颜色对话框对
象
    messagebox1.setText("消息提示对话框");    //设置标题
    messagebox1.setMessage("提示消息默认样式");
    messagebox1.open();                      //显示对话框

    MessageBox messagebox2=new MessageBox(shell,SWT.ICON_ERROR); //构造颜色
对话框对象
    messagebox2.setText("错误提示对话框");    //设置标题
    messagebox2.setMessage("错误消息默认样式");
    messagebox2.open();                      //显示对话框

    MessageBox messagebox3=new MessageBox(shell,SWT.ICON_INFORMATION); //构
造颜色对话框对象
    messagebox3.setText("消息提示对话框");    //设置标题
    messagebox3.setMessage("提示消息默认样式");
    messagebox3.open();

    MessageBox messagebox4=new MessageBox(shell,SWT.ICON_INFORMATION); //构
造颜色对话框对象
    messagebox4.setText("消息提示对话框");    //设置标题
    messagebox4.setMessage("提示消息默认样式");
    messagebox4.open();

    MessageBox messagebox5=new MessageBox(shell,SWT.ICON_QUESTION); //构造
颜色对话框对象
    messagebox5.setText("问题消息提示对话框"); //设置标题
    messagebox5.setMessage("问题提示消息默认样式");
    messagebox5.open();

    shell.open();
    while (!shell.isDisposed())              //检验窗体是非关闭
    {
        if (!display.readAndDispatch())      //检验 Display 线程状态是非忙
        display.sleep ();                    //Display 类线程休眠
    }
}

```

```
    }
    display.dispose ();           //注销 Display 对象资源
}
}
```

显示效果



4.18. ColorDialog 组件

颜色对话框对象 在文本编程中，运用的比较多，用系统构建的颜色集合对对话框选择颜色 ColorDialog 作为对话框的子类，实现了本地 Color 选择，颜色的实现依赖与底层操作系统获取的颜色显示方案，以及图形图像显示器。

ColorDialog 没有拓展样式,只是继承了 Dialog 的样式

样式	说明
SWT.NONE	默认式样
SWT.APPLICATION_MODAL	系统运转状态（1—16 变化）
SWT.MODELESS	非模态的运转状态（0 定制）
SWT.PRIMARY_MODAL	主要的运转状态（1-15 变化）
SWT.SYSTEM_MODAL	系统的运转状态（1-17 变化）

样式的取值不固定，因此不同样式之间实际产生的样式区别很小

Colordialog 的常用方法

函数	作用
getRGB()/setRGB(RGB rgb)	获取、设置颜色对话框中选择的颜色
open()	获取打开对话框

实例代码

```
package example.e03;
```

```
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.ColorDialog;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class ShowColorDialog {

    public static void main(String[] args) {
        Display display=new Display();        // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);            //设置窗体的位置
        shell.setSize(200,100);                //设置窗体大小
        shell.setText("颜色对话框");            //设置窗体标题

        ColorDialog colordialog=new ColorDialog(shell,SWT.NONE); //构造颜色对话框对象
        colordialog.setText("颜色选择对话框");    //设置标题
        colordialog.open();                      //显示对话框

        shell.open();
        while (!shell.isDisposed())            //检验窗体是非关闭
        {
            if (!display.readAndDispatch())    //检验 Display 线程状态是非忙
                display.sleep ();              //Display 类线程休眠
        }
        display.dispose ();                    //注销 Display 对象资源
    }
}
```

显示效果



4.19. FontDialog 组件

字体对话框对象 在文本编程中，运用的比较多，用系统构建的字体集合对对话框选择字体 FontDialog 作为对话框的子类，实现了本地 font 选择，

FontDialog 没有拓展样式,只是继承了 Dialog 的样式

样式	说明
SWT.NONE	默认式样
SWT.APPLICATION_MODAL	系统运转状态（1—16 变化）
SWT.MODELESS	非模态的运转状态（0 定制）
SWT.PRIMARY_MODAL	主要的运转状态（1-15 变化）
SWT.SYSTEM_MODAL	系统的运转状态（1-17 变化）

样式的取值不固定，因此不同样式之间实际产生的样式区别很小

Fontdialog 的常用方法

函数	作用
getFont ()/setFont (Font font)	获取、设置字体对话框中选择的字体
open()	获取打开对话框

实例代码

```
package example.e03;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.FontDialog;
import org.eclipse.swt.widgets.Shell;
```

```

public class ShowFontDialog {

    public static void main(String[] args) {
        Display display=new Display();           // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);              //设置窗体的位置
        shell.setSize(200,100);                  //设置窗体大小
        shell.setText("字体对话框");             //设置窗体标题

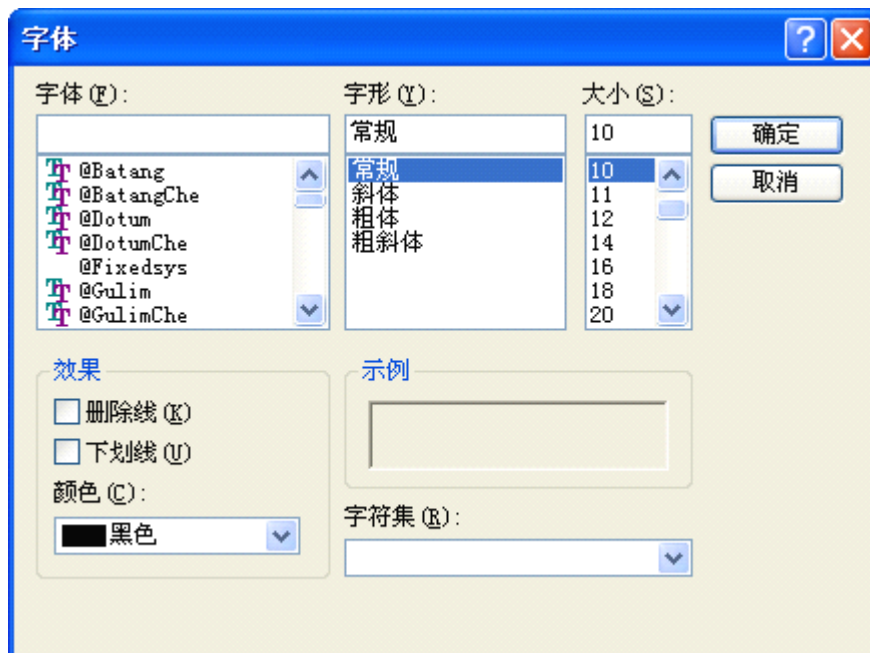
        FontDialog fontdialog=new FontDialog(shell,SWT.NONE); //构造字体选择对话框对
象

        fontdialog.setText("字体选择对话框");    //设置标题
        fontdialog.open(); //显示对话框

        shell.open();
        while (!shell.isDisposed())             //检验窗体是非关闭
        {
            if (!display.readAndDispatch())      //检验 Display 线程状态是非忙
                display.sleep ();                //Display 类线程休眠
        }
        display.dispose ();                      //注销 Display 对象资源
    }
}

```

显示效果



4.20. DirectoryDialog 组件

目录对话框对象 在文件编程中，运用的比较多，用系统构建的文件集合对对话框选择目录 DirectoryDialog 作为对话框的子类，实现了本地目录选择。

DirectoryDialog 的样式没有拓展样式,只是继承了 Dialog 的样式

样式	说明
SWT.NONE	默认式样
SWT.APPLECAUTION_MODAL	系统运转状态（1—16 变化）
SWT.MODELESS	非模态的运转状态（0 定制）
SWT.PRIMARY_MODAL	主要的运转状态（1-15 变化）
SWT.SYSTEM_MODAL	系统的运转状态（1-17 变化）

样式的取值不固定，因此不同样式之间实际产生的样式区别很小

DirectoryDialog 的常用方法

函数	作用
getFilterPath() / setFilterPath (String string)	获取、设置目录对话框中过滤路径
open()	获取打开对话框

实例代码

```
package example.e03;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.DirectoryDialog;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class ShowDirectoryDialog {

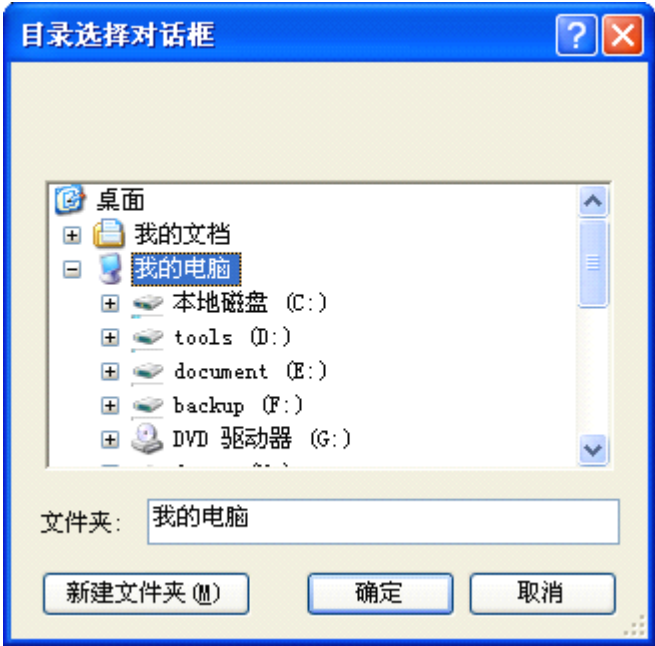
    public static void main(String[] args) {
        Display display=new Display();           // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);              //设置窗体的位置
        shell.setSize(200,100);                  //设置窗体大小
        shell.setText("目录对话框");             //设置窗体标题

        DirectoryDialog directorydialog=new DirectoryDialog(shell,SWT.NONE); //构造目录
        选择对话框对象
        directorydialog.setText("目录选择对话框"); //设置标题
        directorydialog.open(); //显示对话框
    }
}
```

```
shell.open();
while (!shell.isDisposed())           //检验窗体是非关闭
{
    if (!display.readAndDispatch())    //检验 Display 线程状态是非忙
        display.sleep ();             //Display 类线程休眠
}
display.dispose ();                   //注销 Display 对象资源
}
```

运行结果如图



4.21. FileDialog 控件

文件对话框对象 在文件编程中，运用的比较多，用打开对应的文件，选择文件的路径
FileDialog 作为对话框的子类，实现了文件选择。

FileDialog 的样式没有拓展样式,只是继承了 Dialog 的样式

样式	说明
SWT.NONE	默认式样
SWT.APPLICATION_MODAL	
SWT.MODELESS	
SWT.PRIMARY_MODAL	
SWT.SYSTEM_MODAL	

FileDialog 的常用方法

函数	作用
getFilterPath() / setFilterPath (String string)	获取、设置目录对话框中过滤路径
open()	获取打开对话框

实例代码

```

package example.e03;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.swt.widgets.Shell;

public class ShowFileDialog {

    public static void main(String[] args) {
        Display display=new Display();           // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);              //设置窗体的位置
        shell.setSize(200,100);                  //设置窗体大小
        shell.setText("文件对话框");              //设置窗体标题

        FileDialog save_dialog=new FileDialog(shell,SWT.SAVE); //构造文件对话框对象 模
        式 “保持”
        save_dialog.setText("保存文件对话框");    //设置标题
        save_dialog.open();                      //显示对话框

        FileDialog open_dialog=new FileDialog(shell,SWT.OPEN); //构造文件对话框对象 模
        式 “打开”
        open_dialog.setText("打开文件对话框");    //设置标题
        open_dialog.open();

        shell.open();
        while (!shell.isDisposed())              //检验窗体是非关闭
        {
            if (!display.readAndDispatch())      //检验 Display 线程状态是非忙
                display.sleep ();                //Display 类线程休眠
        }
        display.dispose ();                      //注销 Display 对象资源
    }
}

```

4.22. PrintDialog 控件

文件对话框对象 在文件编程中，运用的比较多，用打印的文件，选择打印机，设置属性 `printerDialog` 作为对话框的子类，实现了答应参数。

`printerDialog` 的样式没有拓展样式,只是继承了 `Dialog` 的样式

样式	说明
SWT.NONE	默认式样
SWT.APPLECAUTION_MODAL	
SWT.MODELESS	
SWT.PRIMARY_MODAL	
SWT.SYSTEM_MODAL	

`printerDialog` 的常用方法

函数	作用
open()	获取打开对话框
getStartPage() / setStartPage(int startpage)	获取、设置开始页码
getEndPage() / setEndPage(int endpage)	获取、设置结束页码
getPrintToFile()	是非输出到文件
setScope(int scope)	设置打印的域

`PrinterData` 封装了 `printerDialog` 传递的数据，提供下面是常用字段

字段	作用
ALL_PAGES	页面数
collate	校验对照
copyCount	打印副本数量
driver	打印驱动
endPage	结束页
fileName	打印文件
name	打印机名称
otherData	
PAGE_RANGE	打印起始范围
printToFile	是非输出文件
scope	打印域
startPage	开始页

`Printer` 类封装了打印任务事务

函数	作用
computeTrim(int x, int y, int width, int height)	计算打印区域面积
startPage()	对应打印 data 开始页
endPage()	对应打印 data 结束页

startJob (String jobName)	开始打印任务
cancelJob ()	取消任务
endJob ()	结束任务

实例代码

```

package example.e03;

import org.eclipse.swt.SWT;
import org.eclipse.swt.printing.PrintDialog;
import org.eclipse.swt.printing.Printer;
import org.eclipse.swt.printing.PrinterData;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class ShowPrintDialog {

    public static void main(String[] args) {
        Display display=new Display();           // 创建 Display 类对象

        Shell shell=new Shell(display,SWT.NO);    //建立 shell 主窗体，风格
        shell.setLocation(200, 200);              //设置窗体的位置
        shell.setSize(200,100);                  //设置窗体大小
        shell.setText("颜色对话框");             //设置窗体标题

        PrintDialog printerdialog=new PrintDialog(shell,SWT.NONE); //构造颜色对话框对象
        printerdialog.setText("颜色选择对话框"); //设置标题
        PrinterData printerdata=printerdialog.open(); //显示对话框
        printerdata.fileName="D://test.doc";
        Printer printer=new Printer(printerdata);
        printer.startJob("开始打印");
        printer.endJob();
        shell.open();
        while (!shell.isDisposed())             //检验窗体是非关闭
        {
            if (!display.readAndDispatch())      //检验 Display 线程状态是非忙
                display.sleep ();                //Display 类线程休眠
        }
        display.dispose ();                      //注销 Display 对象资源
    }
}

```

4.23. CCombo 组件

在 SWT 的组件中，有一类组件是可供用户自定义的，在 `org.eclipse.swt.custom` 下面，这类组件是依赖操作系统的，例如 `CCombo` 与 `Combo` 不同的是，它不是系统的 `native` 控件，使用方法可能与 `Combo` 并不二致。`CCombo` 主要是为了超越 `Combo` 的一些限制，实现了一些自定义的方法、例如在 `win32` 中 `combo` 是不可以设置高度的，但是 `CCombo` 则可以，

下拉框（`CCombo` 类）的式样

样式	说明
<code>SWT.NONE</code>	默认式样
<code>SWT.READ_ONLY</code>	只读
<code>SWT.SIMPLE</code>	带水平滚动条

下拉框（`CCombo` 类）的常用方法

函数	作用
add (String string)	在 <code>Combo</code> 中增加一项。
add (String string,int index)	在 <code>Combo</code> 的第 <code>index</code> 项后插入一项
deselectAll ()	使 <code>Combo</code> 组件中的当前选择项置空
removeAll ()	将 <code>Combo</code> 中的所有选项清空
setItems (String[] items)	将数组中的各项依次加入到 <code>Combo</code> 中
select (int index)	将 <code>Combo</code> 的第 <code>index+1</code> 项设为当前选择项
addListener (int Eventtype , Listener lsterer)	设置响应监听事件

实例代码

```
package example.e05;

import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.CCombo;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class ShowCCombo {
    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

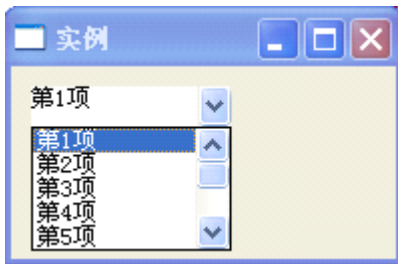
        Shell shell = new Shell(display); //建立 shell 主窗体，风格
        shell.setLocation(200, 200); //设置窗体的位置
        shell.setSize(200, 100); //设置窗体大小
        shell.setText("实例"); //设置窗体标题

        CCombo ccombo = new CCombo(shell, SWT.MULTI | SWT.V_SCROLL); //设置 Combo 下
```


拉框对象

```
ccombo.setLocation(10, 10); //设置下拉框的位置
ccombo.setSize(100, 20); //设置组件大小 对比 Combo 里设置高度是无效的。
ccombo.setTextLimit(20); //设置文本长度上限
ccombo.removeAll(); //清空数据
for (int i = 0; i < 10; i++) {
    ccombo.add("第" + (i + 1) + "项"); //新增数据项
}
ccombo.select(0);
shell.open(); //显示窗体
while (!shell.isDisposed()) //检验窗体是非关闭
{
    if (!display.readAndDispatch()) //检验 Display 线程状态是非忙
        display.sleep(); //Display 类线程休眠
}
display.dispose(); //注销 Display 对象资源
}
```

运行效果



4.24. TableEditor 组件

TableEditor 实现了 Table 表格中的 TableItem 数据可编辑的，以便用户可以编辑该单元格的内容，在 Table 中的数据只是可读的，在 TableEditor 中，TableItem 可以关联其他 Drawable 组件，作为编辑单元。

对于 TableEditor 组件而言，样式单一

样式	说明
SWT.NONE	默认式样

可编辑表格（TableEditor 类）的常用方法

函数	作用
getItem (int index)	获取 index 对应的 trayitem
getColumns ()	获取所有列对象
getColumn (int index)	获取 index 对应的单列对象
getColumnCount ()	获取总列数

getItems()/setItem (TableItem item)	获取、设置行数对象
setEditor (Control editor, TableItem item, int column)	设置编辑的组件 editor 关联的 tableitem,以及列数

实例代码

```
package example.e05;

import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.TableEditor;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.events.SelectionListener;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Table;
import org.eclipse.swt.widgets.TableColumn;
import org.eclipse.swt.widgets.TableItem;
import org.eclipse.swt.widgets.Text;

public class ShowTableEditor {
    public static void main(String[] args) {

        Display display = new Display(); // 创建 Display 类对象

        Shell shell = new Shell(display); // 建立 shell 主窗体，风格
        shell.setLocation(200, 200); // 设置窗体的位置
        shell.setSize(400, 300); // 设置窗体大小
        shell.setText("实例"); // 设置窗体标题

        final Table table = new Table(shell, SWT.FULL_SELECTION);
        table.setLocation(10, 10); // 设置表格的位置
        table.setHeaderVisible(true); // 设置列标题
        table.setBounds(10, 10, 239, 256);

        final TableEditor editor = new TableEditor(table);
        editor.horizontalAlignment = SWT.RIGHT;
        editor.grabHorizontal = true;

        TableColumn tablecloumn1 = new TableColumn(table, SWT.LEFT); // 构造表单列对象，设置样式左对齐
        tablecloumn1.setText("第一列"); //设置列标题
        tablecloumn1.setWidth(80); //设置宽度 默认宽度为 0（通常为不可见）

        TableColumn tablecloumn2 = new TableColumn(table, SWT.LEFT); // 构造表单列对象，设置样式 右对齐
```

```

tablecloumn2.setText("第二列");
tablecloumn2.setWidth(80);

TableColumn tablecloumn3 = new TableColumn(table, SWT.LEFT); // 构造表单列对象，设置样式 右对齐
tablecloumn3.setText("第三列");
tablecloumn3.setWidth(80);

TableItem tableitem1 = new TableItem(table, SWT.NONE); //构造数据值对象(行对象)
tableitem1.setText(new String[] { "1", "皇家马德里", "12321312" }); //填充数据项的值
TableItem tableitem2 = new TableItem(table, SWT.NONE); //构造数据值对象(行对象)
tableitem2.setText(new String[] { "2", "巴萨", "15462312" }); //填充数据项的值
TableItem tableitem3 = new TableItem(table, SWT.NONE); //构造数据值对象(行对象)
tableitem3.setText(new String[] { "3", "尤文图斯", "12321312" }); //填充数据项的值
TableItem tableitem4 = new TableItem(table, SWT.NONE); //构造数据值对象(行对象)
tableitem4.setText(new String[] { "4", "国际米兰", "12321312" }); //填充数据项的值

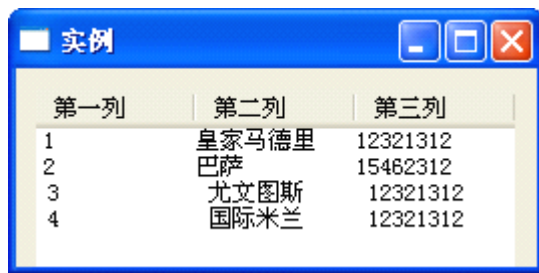
table.addSelectionListener(new SelectionListener() {
    public void widgetDefaultSelected(SelectionEvent e) {
        widgetSelected(e);
    }
    public void widgetSelected(SelectionEvent e) {
        int index = table.getSelectionIndex(); //获取 Select 行
        int colcount=table.getColumnCount(); //获取列数
        TableItem tableitem = table.getItem(index); //得到 select 的行 TableItem
        Text text=null ;
        for (int i = 0; i < colcount; i++) {

            text = new Text(table, SWT.NONE); //初始化一个可编辑组件 Text
            text.setText(tableitem.getText(i)); //获取表格每一行中的数据
            editor.setEditor(text); //设置 table 的组件
            editor.setItem(tableitem); //关联到对应的 TableItem
            editor.setColumn(i); //设置编辑列
        };
    }
});
shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}
display.dispose(); // 注销 Display 对象资源
}

```

```
}
```

运行结果如图



第一列	第二列	第三列
1	皇家马德里	12321312
2	巴萨	15482312
3	尤文图斯	12321312
4	国际米兰	12321312

5 SWT 的组件布局

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一个网格。

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。

5.1. 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
```

5.2. 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 setData()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);
```

5.3. 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 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。

2. 布局数 类（`GridData` 类）

`GridData` 是 `GridLayout` 专用的布局数 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。

5.4. 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 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 ，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

①FormData 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。

②FormData 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下 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（ $a=m/n$, $n \neq 0$ ），b 是偏移量，沿 x、y 轴正方向的偏移量为正，反之为负。

①FormAttachment 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（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（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（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。

6 SWT 事件监听机制

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。

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。

在 SWT 中的任意组件都可以添加监听事件，能够组合两种或者两种以上的事件响应。通用的方法是

```
Control.addListener(EventType, new Listener(){
```

```
    public void handleEvent(Event arg0) {
        //实现对应的逻辑
    }
```



```

    }
});

```

这样的话, 对于一个组件可能必需写多个类似的监听器, 这样代码的维护工作量将是巨大的, 因为通常你只能根据 `EventType`, 来识别响应的监听事件, 而代码按照这种通用格式, 势必会产生许多结构相同的代码块, 因此建议采取下列这种分类的监听事件的写法, 对应的 `Event` 都是继承父类 `Event`, 响应的 `Listener` 都是继承 `SWTEventListener`, 这样分类管理就清晰可见。

对应的事件有

事件源	监听器	适配器	处理事件方法(Listener)
SelectionEvent(组件被选择)	SelectionListener	SelectionAdapter	widgetSelected(SelectionEvent e)
ControlEvent (组件移动, 大小变化)	ControlListener	ControlAdapter	controlMoved(ControlEvent e) controlResized(ControlEvent e)
MouseEvent (鼠标事件)	MouseListener	MouseAdapter	mouseDoubleClick(MouseEvent e) mouseDown(MouseEvent e) mouseUp(MouseEvent e)
	MouseTrackListener	MouseTrackAdapter	mouseEnter(MouseEvent e) mouseExit(MouseEvent e) mouseHover(MouseEvent e)
	MouseMoveListener		mouseMove(MouseEvent e)
KeyEvent (键盘事件)	KeyListener	KeyAdapter	keyPressed(KeyEvent e) keyReleased(KeyEvent e)
FocusEvent (焦点事件)	FocusListener	FocusAdapter	focusGained(FocusEvent e) focusLost(FocusEvent e)
ArmEvent 菜单拖动 (滑动) 事件	ArmListener	ArmAdapter	widgetArmed(ArmEvent e)
DisposeEvent (组件注销事件)	DisposeListener	DisposeAdapter	widgetDisposed(DisposeEvent e)
HelpEvent (帮助信息事件)	HelpListener	HelpAdapter	helpRequested(HelpEvent e)
MenuEvent (菜单事件)	MenuListener	MenuAdapter	menuHidden(MenuEvent e) menuShown(MenuEvent e)
ModifyEvent (文本修改事件)	ModifyListener	ModifyAdapter	modifyText(ModifyEvent e)
PaintEvent (组件绘制事件)	paintListener		paintControl(PaintEvent e)
ShellEvent (窗口事件)	ShellListener	ShellAdapter	shellActivated(ShellEvent e) shellClosed(ShellEvent e) shellDeactivated(ShellEvent e)

			shellIconified(ShellEvent e)
TraverseEvent (焦点移动事件)	TraverseListener		keyTraversed(TraverseEvent e)
TreeEvent (树变化事件)	TreeListener	TreeAdapter	treeCollapsed(TreeEvent e) treeExpanded(TreeEvent e)
VerifyEvent (校验事件)	VerifyListener		verifyText(VerifyEvent e)

在这些时间中，有些事件只是针对特定的组件，像 TreeEvent 这种事件只是针对 Tree 的结构伸展的处理，同时从 SWT 的 Event 的继承关系可以看出，TreeEvent 本质上是一种 SelectionEvent，因此上述所列的 Event 之间的关系并不是并列的关系，而是存在一种继承关系的，因此运用的时候，需要审查该用什么事件去响应对应的操作最合适。

下面将举个几个例子感受一些 SWT 的事件处理机制

首先举一个按钮的例子，通常点击按钮是用的最频繁的工作

```
package example.e04;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.MouseEvent;
import org.eclipse.swt.events.MouseListener;
import org.eclipse.swt.events.MouseMoveListener;
import org.eclipse.swt.events.MouseTrackListener;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class ButtonEvent {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

        Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体，风格
        shell.setLocation(200, 200); // 设置窗体的位置
        shell.setSize(200, 100); // 设置窗体大小
        shell.setText("按钮实例"); // 设置窗体标题

        Button button = new Button(shell, SWT.NO); // 构造 Button 对象，设置样式
        button.setLocation(0, 0); // 设置 Button 的位置
        button.setSize(80, 60); // 设置 Button 大小
        button.setText("Button 实例"); // 设置 Button 标题
        button.addMouseListener(new MouseListener() {
            //e.button    1 代表左键    2 代表 中键    3 代表 右键
            public void mouseDoubleClick(MouseEvent e) {
```

```
        if (e.button == 1) {
            System.out.println("鼠标左键双击");
        }
        if (e.button == 2) {
            System.out.println("鼠标中键双击");
        }
        if (e.button == 3) {
            System.out.println("鼠标右键双击");
        }
    }
    public void mouseDown(MouseEvent e) {
        if (e.button == 1) {
            System.out.println("鼠标左键按下");
        }
        if (e.button == 2) {
            System.out.println("鼠标中键按下");
        }
        if (e.button == 3) {
            System.out.println("鼠标右键按下");
        }
    }
    public void mouseUp(MouseEvent e) {
        if (e.button == 1) {
            System.out.println("鼠标左键松开");
        }
        if (e.button == 2) {
            System.out.println("鼠标中键松开");
        }
        if (e.button == 3) {
            System.out.println("鼠标右键松开");
        }
    }
});

button.addMouseListener(new MouseMoveListener() {

    public void mouseMove(MouseEvent e) {
        System.out.println("鼠标的的位置 X: " + e.x + "Y:" + e.y);
    }
});

button.addMouseTrackListener(new MouseTrackListener() {

    public void mouseEnter(MouseEvent arg0) {
```

```

        System.out.println("鼠标进入 button 区域");
    }
    public void mouseExit(MouseEvent arg0) {
        System.out.println("鼠标离开 button 区域");
    }
    public void mouseHover(MouseEvent arg0) {
        System.out.println("鼠标停留 button 上空区域");
    }
});
shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}
display.dispose(); // 注销 Display 对象资源
}
}

```

下面介绍文本框中的事件

```

package example.e04;

import java.util.Vector;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.KeyEvent;
import org.eclipse.swt.events.KeyListener;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.events.VerifyEvent;
import org.eclipse.swt.events.VerifyListener;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;

public class TextEvent {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

        Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体，风格
        shell.setLocation(200, 200); // 设置窗体的位置
        shell.setSize(200, 100); // 设置窗体大小
    }
}

```

```
shell.setText("文本实例"); // 设置窗体标题

Text text = new Text(shell, SWT.NO); //构造 Text 对象, 设置样式
text.setLocation(50, 0); //设置 Text 的位置
text.setSize(100, 50); //设置 Text 大小
text.setText(""); //设置 Text 标题

text.addModifyListener(new ModifyListener() {

    public void modifyText(ModifyEvent arg0) {

        System.out.println("文本发生修改");
    }

});

text.addKeyListener(new KeyListener() {

    public void keyPressed(KeyEvent e) {
        System.out.println("键盘的" + e.character + "别按下");
    }

    public void keyReleased(KeyEvent e) {
        System.out.println("键盘的" + e.character + "松开");
    }

});

text.addVerifyListener(new VerifyListener() {
    Vector<String> v = new Vector<String>(10);

    public void verifyText(VerifyEvent e) {
        v.add(e.text);
        if (v.size() < 10) {
            System.out.println("数据长度小于 10 char");
        }
    }
});

shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}
```

```
display.dispose(); // 注销 Display 对象资源

    }

}
```

接着介绍 Tree 的组件，相应的 Menu Toolbar Table 等组件都可以类比

```
package example.e04;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.TreeListener;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Tree;
import org.eclipse.swt.widgets.TreeItem;

public class TreeEvent {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

        Shell shell = new Shell(display, SWT.NO); // 建立 shell 主窗体，风格
        shell.setLocation(200, 200); // 设置窗体的位置
        shell.setSize(400, 300); // 设置窗体大小
        shell.setText("tree 实例"); // 设置窗体标题

        Tree tree = new Tree(shell, SWT.NO); // 构造 Tree 对象，设置样式
        tree.setLocation(0, 0); // 设置 Tree 的位置
        tree.setBounds(0, 0, 102, 190);
        tree.addTreeListener(new TreeListener() {

            public void treeCollapsed(org.eclipse.swt.events.TreeEvent arg0) {
                System.out.println("节点收拢");
            }

            public void treeExpanded(org.eclipse.swt.events.TreeEvent arg0) {
                System.out.println("节点展开");
            }

        });

        TreeItem treeitem1 = new TreeItem(tree, SWT.NO); // 构造 TreeItem 对象，设置样式
        treeitem1.setText("树的一课子树");
    }
}
```

```

TreeItem treeitem2 = new TreeItem(tree, SWT.NO); // 构造 TreeItem 对象，设置样式
treeitem2.setText("树的二课子树");

TreeItem treeitem3 = new TreeItem(treeitem1, SWT.NO); // 构造 TreeItem 对象，设置
样式
treeitem3.setText("树的分支");
TreeItem treeitem4 = new TreeItem(treeitem2, SWT.NO); // 构造 TreeItem 对象，设置
样式
treeitem4.setText("树的分支");

shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}
display.dispose(); // 注销 Display 对象资源
}
}

```

最后简述窗口 Shell 的事件

```

package example.e04;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.DisposeEvent;
import org.eclipse.swt.events.DisposeListener;
import org.eclipse.swt.events.ShellListener;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.MessageBox;
import org.eclipse.swt.widgets.Shell;

public class ShellEvent {

    public static void main(String[] args) {
        Display display = new Display(); // 创建 Display 类对象

        final Shell shell = new Shell(display); // 建立 shell 主窗体，风格
        shell.setLocation(200, 200); // 设置窗体的位置
        shell.setSize(200, 100); // 设置窗体大小
        shell.setText("实例"); // 设置窗体标题
    }
}

```

```

shell.addShellListener(new ShellListener() {

    public void shellActivated(org.eclipse.swt.events.ShellEvent arg0) {
        System.out.println("窗口激活");
    }
    public void shellClosed(org.eclipse.swt.events.ShellEvent arg0) {
        System.out.println("窗口关闭");
    }
    public void shellDeactivated(org.eclipse.swt.events.ShellEvent arg0) {
        System.out.println("窗口未激活(失去焦点)");
    }
    public void shellDeiconified(org.eclipse.swt.events.ShellEvent arg0) {
        System.out.println("窗口非最小化");
    }
    public void shellIconified(org.eclipse.swt.events.ShellEvent arg0) {
        System.out.println("窗口最小化");
    }
});

shell.addDisposeListener(new DisposeListener() {

    public void widgetDisposed(DisposeEvent arg0) {
        MessageBox confrim = new MessageBox(shell, SWT.ICON_QUESTION
            | SWT.YES | SWT.NO);
        confrim.setMessage("是否关闭窗口?");
        confrim.setText("消息提示");
        if (confrim.open() == SWT.YES) {
            shell.dispose();
        }
    }
});
shell.open(); // 显示窗体
while (!shell.isDisposed()) // 检验窗体是非关闭
{
    if (!display.readAndDispatch()) // 检验 Display 线程状态是非忙
        display.sleep(); // Display 类线程休眠
}
display.dispose(); // 注销 Display 对象资源
}
}

```

但是从来没有一种方法是能够解决所有问题的，对应的处理方法处理对应的问题，用哪种方法写完全取决于实际问题的场景，例如，当处理响应时间中，只有 `Mousedown Mouseup` 的时候，最好的办法就是 `addMouseListener`，如果当事件关联较多是就不适合用单一的事件模

型，而建议使用通用机制，组合响应事件，例如 在 Mousedown Mouseup 还有 mousemove 的时候，建议 addListener (int event ,listener)
既能保证响应的事件能够处理，又能实现响应事件之间能通讯。

7 应用 SWT 绘制 2D 图像图像

在 SWT 中完成图像图像绘制，和 Device 打交道是 GC 类（Graphics Context）
我们可以在任何实现了 org.eclipse.swt.graphics.Drawable 接口的类上绘制图形，这包括一个控件，一幅图像，一个显示设备或一个打印设备。类 org.eclipse.swt.graphics.GC 是一个封装了所有可执行的绘图操作的图形上下文（Graphics Context）。

GC 的构造

GC() 构造一个默认的 GC

GC(Drawable drawable) 构造一个实现了 Drawable 接口的组件

GC(Drawable drawable, int style) 构造一个实现了 Drawable 接口的组件，并设置样式

GC 的常用的方法

函数	说明
drawLine (int x1,int y1, int x2, int y2)	画线，定义起点 (x1,y1)，终点坐标(x2,y2)
drawArc (int x,int y,int width, int height,int startAngle,int arcAngle)	画弧线 弧中心 (x,y)
drawBitmap (Image srcImage, int srcX, int srcY, int srcWidth, int srcHeight, int destX, int destY, int destWidth, int destHeight, boolean simple)	画 bitmap ，原 image 文件 srcImage，
drawIcon (Image srcImage, int srcX, int srcY, int srcWidth, int srcHeight, int destX, int destY, int destWidth, int destHeight, boolean simple)	画 icon ，原 icon 文件 srcImage，
drawOval (int x, int y, int width, int height)	画椭圆 圆心 (x,y) 长轴 width, 短轴 height
drawPath (Path path)	绘制路径
drawPoint (int x, int y)	画点 (x,y)
drawPolygon (int[] pointArray)	绘制多边形，多个点相连，封闭
drawPolyline (int[] pointArray)	绘制多边形，多个点相连，不许封闭
drawRectangle (Rectangle rect)	画矩形
drawRoundRectangle (int x, int y, int width, int height, int arcWidth, int arcHeight)	画圆角矩形
drawRoundRectangleGdip (int gdipGraphics, int brush, int x, int y, int width, int height, int arcWidth, int arcHeight)	画圆角矩形，圆角边设置样式 gdipG

	raphics
drawString (java.lang.String string,int x, int y)	绘制字符串
drawText (java.lang.String string, int x, int y)	绘制文本
fillArc (int x, int y, int width, int height, int startAngle, int arcAngle)	填充弧形
fillGradientRectangle (int x, int y, int width, int height, boolean vertical)	填充不规则矩形区域
fillOval (int x, int y, int width, int height)	填充多边形
fillPath (Path path)	填充路径
fillPolygon (int[] pointArray)	填充多边形
fillRectangle (int x,int y,int width, int height)	填充矩形
fillRectangle (Rectangle rect)	填充矩形
fillRoundRectangle (int x, int y, int width, int height, int arcWidth, int arcHeight)	填充圆角矩形
fillRoundRectangleGdip (int gdipGraphics,int brush,int x, int y,int width,int height,int arcWidth,int arcHeight)	填充圆角矩形，圆角边设置样式 gdipGraphics
getAdvanced ()/setAdvanced(boolean advanced)	设置 GC 设备级别
getAdvanceWidth (char ch)	
getAlpha ()/setAlpha(int alpha)	获取、设置混和色
getAntialias ()/setAntialias(int antialias)	获取、设置属性反锯齿开关
getBackground ()/setBackground(Color color)	获取、设置背景色
getBackgroundPattern ()/setBackgroundPattern(Pattern pattern)	获取、设置背景模式
getCharWidth (char ch)	
getClipping ()/setClipping(int clipRgn)	获取、设置剪切域
getFillRule ()/setFillRule(int rule)	获取、设置填充规则
getFont ()/setFont(Font font)	获取、设置字体
getFontMetrics ()	获取、设置光标字体规律，文本编程
getForeground ()/setForeground(Color color)	获取、设置前置色
getGCData ()	获取 GC 数据
getInterpolation ()/setInterpolation(int interpolation)	获取、设置涂改属性
getLineCap ()/setLineCap(int cap)	设置线条覆盖样式
getLineDash ()/setLineDash(int[] dashes)	设置折线样式
getLineJoin ()/setLineJoin(int join)	设置连线的节点样式
getLineStyle ()/setLineStyle(int lineStyle)	设置线条的样式
getLineWidth ()/setLineWidth(int lineWidth)	设置线条宽度
getStyle ()	获取组件 GC 样式

getTextAntialias()/setTextAntialias (int antialias)	设置文本的平滑度属性
getTransform (Transform transform)/ setTransform (Transform transform)	设置变换属性
measureSpace (int font, int format)	度量空间区域
setPen (int newColor,int newWidth,int lineStyle,int capStyle, int joinStyle,int[] dashes)	设置画笔属性

绘制线条

```
package example.e06;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.GC;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Shell;

public class ShowDrawLine {

    public static void main(String[] args) {
        Display display = new Display();
        final Shell shell = new Shell(display);
        Listener listener = new Listener() {
            int startX = 0;
            int startY = 0;

            public void handleEvent(Event event)
            {
                switch (event.type)
                {
                    case SWT.MouseMove :
                        if ((event.stateMask & SWT.BUTTON1) == 0)
                            break; // 判断是否为鼠标左键，如果不是跳出
                        GC gc = new GC(shell);
                        gc.drawLine(startX, startY, event.x, event.y);
                        gc.dispose();

                    case SWT.MouseDown :
                        startX = event.x;
                        startY = event.y;
                        break;
                }
            }
        };
    }
}
```

```

        shell.addListener(SWT.MouseDown, listener);
        shell.addListener(SWT.MouseMove, listener);
        shell.open();
        while (!shell.isDisposed())
        {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}

```

完成

```

package example.e06;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class ShowDraw {

    public ShowDraw() {
        Display display = new Display(); // 创建 Display 类对象

        final Shell shell = new Shell(display, SWT.NO); //建立 shell 主窗体，风格
        shell.setLocation(200, 200); //设置窗体的位置
        shell.setSize(800, 600); //设置窗体大小
        shell.setText("Hello World"); //设置窗体标题

        MouseDraw md = new MouseDraw(shell);    //将鼠标操作封装到 MouseDraw 中

        shell.addMouseListener(md);            //增加响应事件
        shell.open(); //显示窗体
        while (!shell.isDisposed()) //检验窗体是非关闭
        {
            if (!display.readAndDispatch()) //检验 Display 线程状态是非忙
                display.sleep(); //Display 类线程休眠
        }
        display.dispose(); //注销 Display 对象资源
    }

    public static void main(String[] args) {
        new ShowDraw();
    }
}

```

```
}  
}
```

MouseDown 封装的实现类

```
package example.e06;  
  
import java.util.logging.Logger;  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.events.MouseEvent;  
import org.eclipse.swt.events.MouseListener;  
import org.eclipse.swt.graphics.GC;  
import org.eclipse.swt.graphics.Point;  
import org.eclipse.swt.widgets.Composite;  
  
public class MouseDraw implements MouseListener {  
    private Point startP;  
    private Point endP;  
    private Composite composite;  
    private Logger logger=Logger.getLogger("MouseDown");  
  
    public MouseDraw(Composite composite) {  
        this.composite=composite;  
    }  
    public void drawLine(Composite composite,Point s ,Point d)  
    {  
        GC gc=new GC(composite);  
        gc.setAntialias(SWT.ON);  
        gc.drawLine(s.x, s.y,d.x, d.y);  
        gc.drawRectangle(s.x, s.y, d.x-s.x, d.y-s.y);  
        gc.dispose();  
    }  
    public void mouseDoubleClick(MouseEvent e) {  
  
    }  
  
    public void mouseDown(MouseEvent e) {  
        startP=new Point(e.x,e.y);  
        logger.info("起点"+startP.toString());  
    }  
  
    public void mouseUp(MouseEvent e) {  
        endP=new Point(e.x,e.y);  
    }  
}
```

```
        logger.info("结束点"+endP.toString());
        drawLine(composite, startP, endP);
    }
}
```

8 SWT 的 OpenGL 应用

相较于 Java3D API 来说，SWT 以前在 3D 图形绘制方面一直没有什么好的表现。OpenGL 的加入会不会使 SWT 在 3D 领域有所作为还尚未可知，不过起码 IBM 的程序员们给了 SWT 机会。当大家了解了这个正处于试验阶段的组合之后，我们在 SWT 上绘制 3D 图形就不再是噩梦。

OpenGL 是一个为创建高性能 2D，3D 图形而设计的多平台的标准。其硬件和软件的实现存在于多个系统之中，包括 Windows，Linux 和 MacOS。OpenGL 可以用于渲染简单的 2D 图形或复杂的 3D 游戏图形（OpenGL 最主要的应用领域就是游戏）。作为一个正在处于事件阶段的 Eclipse 插件，我将在下面的小节中介绍如何在 SWT 窗口组件上用 SWT 绘制图形。在 Eclipse 最新的 3.2 版中，对 OpenGL 的支持被集成到 org.eclipse.swt 项目中，所以我们在实现的时候即可以选择以插件方式进行，也可以直接利用已经集成好的组件来进行图形操作。在本节，我们将以插件方式为例对代码进行说明。

SWT OpenGL 插件

SWT 实现了 OpenGL1.1 全部功能。包括三个核心类和一个数据类。核心类为 GLContext，GL 和 GLU。GLContext 架起了 SWT 和 OpenGL 之间的桥接。一个 Context 必须用 Drawable，通常是用 Canvas 来创建，OpenGL 可以在 Drawable 上渲染场景。需要注意的是，当 context 不再被使用的时候就应该将它释放掉。同样，一旦某个 context 被释放掉之后，就不应该再次试图去渲染它。每次 Drawable 改变大小的时候，context 都需要通过调用其 resize 方法在通知这一事件。这个方法的调用让 context 调整自己的 view port 和视图参数。在下一节中将描述一个处理这一部分任务的类。

当 context 可用的时候，我们就可以通过定义在 GL 和 GLU 的一系列方法调用来绘制场景。一个 GL 类大概有超过 330 条命令。在 GL 和 GLU 中定义的这些函数和他们的 Native 实现几乎是一一对应的。下图给出了一个绘制矩形的例子，我们可以看到用 C 写成的 API 和 SWT

OpenGL API 是何其相似：

(a) C Code

```

void drawScene() {
    glClear(GL_COLOR_BUFFER_BIT
           | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, -5.0f);

    glBegin(GL_TRIANGLES);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
        glVertex3f(0.0f, 1.0f, 0.0f);
    glEnd();

    glutSwapBuffers();
}

```

(b) Java Code

```

public void drawScene() {
    GL.glClear(GL.GL_COLOR_BUFFER_BIT
              | GL.GL_DEPTH_BUFFER_BIT);
    GL.glLoadIdentity();
    GL.glTranslatef(0.0f, 0.0f, -5.0f);

    GL.glBegin(GL.GL_TRIANGLES);
        GL.glVertex3f(-1.0f, -1.0f, 0.0f);
        GL.glVertex3f(1.0f, -1.0f, 0.0f);
        GL.glVertex3f(0.0f, 1.0f, 0.0f);
    GL.glEnd();

    glContext.swapBuffers();
}

```

SWT OpenGL 编程基础

在下面的小节中，我将描述一个显示四幅 3D 图像的应用程序。应用程序采用了 GLSense，这是一个用于显示 OpenGL 场景的工具类。它和 SWT 的 Canvas 很像，所区别的是它所展现的内容是用 OpenGL 命令渲染的，而不是使用 GC 来绘制。要做到这一点，我们需要将一个 GLContext 类和一个 SWT Canvas 相关联，并且无论何时，当前上下文中的内容都应该是由在 drawScene 中定义的命令来渲染的。

```

1 public class GLScene {
2     private GLContext context;
3     private Canvas canvas;
4
5     public GLScene(Composite parent) {
6         this.canvas = new Canvas(parent, SWT.NONE);
7         this.canvas.addListener(new ControlAdapter() {
8             public void controlResized(ControlEvent e) {
9                 resizeScene();
10            }
11        });
12        this.canvas.addDisposeListener(new DisposeListener() {
13            public void widgetDisposed(DisposeEvent e) {
14                dispose();
15            }
16        });
17        this.init();
18        Rectangle clientArea = parent.getClientArea();
19        this.canvas.setSize(clientArea.width, clientArea.height);
20    }

```

在构造函数中，一个 SWT Canvas 被创建出来。这就是那个要和一个 GLContext 相关联的 Canvas 实例。紧接着，这个 Canvas 又注册了两个监听器。第一个监听器的作用是确保这个 Canvas 无论何时被改变大小，其相应的 GLContext 也会收到通知并适当的改变大小。第二个监听器主要用于确保一旦 Canvas 被释放之后，其相对应的 GLContext 的也同时被释放。为了确保渲染区域是一个非零大小的区域，父组件的客户矩形区被取出来用于设置该 Canvas 的初始大小。这个初始大小可以在稍后用布局管理器或用户 Action 来修改。


```
22     protected void resizeScene() {
23         Rectangle rect = this.canvas.getClientArea();
24         this.context.resize(0, 0, rect.width, rect.height);
25     }
26
27     protected void dispose() {
28         if (this.context != null) {
29             this.context.dispose();
30             this.context = null;
31         }
32     }
```

GLScene 将 Canvas 的全部区域用于绘图。无论 Canvas 何时调整其尺寸，我们都要获取客户区并将新的宽度和高度传递给 Context，而 context 将根据新的宽度和高度适当的调整视图。

GLScene 被分割为两个部分：初始化 Context 和初始化 OpenGL 的状态机。对于 Context 来说，我们只是简单的建立一个新的 GLContext 并使它成为当前被使用的 Context。OpenGL 的渲染总是在当前的 context 上进行绘制，因此如果你有超过一个活动的 GLScene，很重要的一点是要在所有绘制动作发生之前将它的 Context 设置为当前的 Context。initGL 方法最开始提供清除颜色缓存颜色，随后建立了一个深度缓存（depth buffer）。第 47 行指出了深度值如何进行比较。这一比较函数主要用于拒绝或接受正在引用的像素。GL.GL_LEQUAL 选项指定接受那些在视图上更接近或有相同距离的像素。第 48 行启动了深度测试（depth test），紧接的一行设定阴影模型为 GL.GL_SMOOTH，这一设定的效果是如果表面上的两个顶点颜色不同的话，系统将对颜色进行插值。最后，第 50 行要求渲染引擎在计算颜色和纹理协调插值运算的时候起到关键的作用。

GLScene 类的最后两个方法用于处理重绘和场景绘制。当场景何时需要重绘的时候，第一个方法为其他类提供重绘操作的接口。第二个方法主要用于让继承 GLScene 的子类覆写。其缺省实现只是简单的清除了颜色和深度缓存，通过装在鉴别矩阵（identify matrix）重新恢复调整系统。

3D Chart

利用上一节的准备，我们已经将主应用程序进行了划分。这个图像显示了 4 组数据。每一组数据都是由相同的固定点所组成，每个点都是从 0.0 到 10.0 之间的一个正值。

示例程序运行在一个非常简单的 Eclipse view 上，唯一值得注意的是 Refresher，这个线程将强迫 OpenGL 场景被周期性的重绘。通过这种方法，当视图被移动或旋转的时候，component 总能进行有效的更新渲染效果。run()方法调用的时间间隔为 100 毫秒，所以理论上的图像速度能达到每秒 10 帧。


```

13 public class Refresher implements Runnable {
14     public static final int DELAY = 100;
15
16     private GLScene scene;
17
18     public Refresher(GLScene scene) {
19         this.scene = scene;
20     }
21
22     public void run() {
23         if (this.scene != null && !this.scene.isDisposed()) {
24             this.scene.render();
25             this.scene.getDisplay().timerExec(DELAY, this);
26         }
27     }
28 }

```

每个数据集合的点的值是用圆柱体来表示的。通过执行 3 个 GLU 调用，我们就能够绘制圆柱体：其中的两个用于渲染圆柱体两头的圆盘部分，另外一个用于渲染圆柱体的四周。例如，要渲染两个单元高的圆柱体，你可以用下面的代码来实现：

```

1 int qobj = GLU.gluNewQuadric();
2 GL.glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
3 GLU.gluDisk(qobj, 0.0, 1.0, 32, 1);
4 GLU.gluCylinder(qobj, 1.0f, 1.0, 2.0, 32, 1);
5 GL.glTranslatef(0.0f, 0.0f, 2.0f);
6 GLU.gluDisk(qobj, 0.0, 1.0, 32, 1);
7 GL.glTranslatef(0.0f, 0.0f, -2.0f);
8 GL.glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
9 GLU.gluDeleteQuadric(qobj);

```



第一行申请了绘制圆盘和圆柱所需的二次曲面。然后整个场景被逆时针旋转了 90 度，以便圆柱体可以被垂直绘制。下一步，底部的圆盘被渲染，然后是圆柱体的四周。在我们能够绘制顶部圆盘的时候，通过场景转换（scene translation），我们可以在 Z 轴移动两个单元。最后一个圆盘随后被绘制出来，调整系统通过向回移动两个单元来进行恢复。最后，由第一行申请的二次曲面被释放掉。

按照上述方法运行程序是很费时间的。当仅绘制一个圆柱体的时候，效率低下不是一个很严重的问题，但如果要绘制成百个对象的话就会严重影响程序的执行性能。对于这种情况，OpenGL 给出了一个解决这个问题的技巧，就是使用显示列表（display list）。

一个显示列表是一组已编译的 OpenGL 命令。定义命令集合的列表被放在 `glNewList(int list, int mode)` 和 `glEndList()` 方法调用之间。第一个参数必须是一个正整数，可以用来唯一的表示一个被创建的显示列表。你可以让 GL 用 `glGenLists(int n)` 方法为你生成多个列表标识符。第二个参数用于指定列表是否被编译或编译之后立即被执行。大多数情况下你都需要编译这个列表。然后，你可以使用 `glCallList(int list)` 方法来显示整个列表。

9 SWT 和 Swing、AWT 技术比较

9.1. AWT 概述

Abstract Windows Toolkit (AWT) 是最原始的 Java GUI 工具包。AWT 的主要优点是，它在 Java 技术的每个版本上都成为了一种标准配置，包括早期的 Web 浏览器中的 Java 实现；另外它也非常稳定。这意味着我们不需要单独安装这个工具包，在任何一个 Java 运行环境中都可以使用它，这一点正是我们所希望的特性。

AWT 是一个非常简单的具有有限 GUI 组件、布局管理器 and 事件的工具包（参见 清单 1、清单 2 和 清单 3）。这是因为 Sun 公司决定为 AWT 使用一种最小公分母（LCD）的方法。因此它只会使用为所有 Java 主机环境定义的 GUI 组件。最终的结果非常不幸，有些经常使用的组件，例如表、树、进度条等，都不支持。对于需要更多组件类型的应用程序来说，我们需要从头开始创建这些组件。这是一个很大的负担。

清单 1. 基本的 AWT Class 树（全部在 java.awt 包中，“*” 表示抽象）

Object	CheckboxGroup	*Component	Button
Canvas	CheckBox	Choice	Container
Panel	Applet	ScrollPane	Window
*Dialog	Frame	Label	List
TextComponent	TextArea	TextField	MenuComponent
MenuItem	CheckboxMenuItem	Menu	PopupMenu

注意：另外几个包中还有其他一些 AWT 组件，但是这是基本的组件集。

清单 2. AWT 提供了下面的布局管理器（全部在 java.awt 包中，“*” 表示接口）

*LayoutManager	FlowLayout
GridLayout	*LayoutManager2
BorderLayout	CardLayout
GridBagLayout	

注意：另外几个包中还有一些 AWT 布局管理器，很多都是为它们进行布局的容器专门定制的，但是这是基本的布局管理器集。

清单 3. AWT 提供了以下事件（大部分在 java.awt.events 包中）

Object EventObject	AWTEvent	ActionEvent	AdjustmentEvent
ComponentEvent	ContainerEvent	FocusEvent	InputEvent
KeyEvent	MouseEvent	MouseWheelEvent	PaintEvent
WindowEvent	HierarchyEvent	InputMethodEvent	InvocationEvent
ItemEvent	TextEvent		

注意：其他几个包中还有另外一些 AWT 事件，但是这是基本的事件集。这些是从更通用

的事件生成的具体事件。

通常对于 AWT 来说（也适用于 Swing 和 SWT），每个事件类型都有一个相关的 XxxListener 接口（XxxAdapter 的实现可能为空），其中 Xxx 是去掉 Event 后缀的事件名（例如，KeyEvent 事件的接口是 KeyListener），用来把事件传递给处理程序。应用程序会为自己感兴趣处理的事件的事件源（GUI 组件或部件）进行注册。有时监听接口要处理多个事件。

AWT 的一个很好的特性是它通常可以对 GUI 组件自动进行销毁。这意味着您几乎不需要对组件进行销毁。一个例外是高级组件，例如对话框和框架。如果您创建了耗费大量主机资源的资源，就需要手动对其进行销毁。

AWT 组件是“线程安全的（thread-safe）”，这意味着我们不需要关心在应用程序中是哪一个线程对 GUI 进行了更新。这个特性可以减少很多 GUI 更新的问题，不过使 AWT GUI 运行的速度更慢了。

AWT 让我们可以以自顶向下（top-down）或自底向上（bottom-up）或以任意组合顺序来构建 GUI。自顶向下的意思是在创建子组件之前首先创建容器组件；自底向上的意思是在创建容器（或父）组件之前创建子组件。在后一种情况中，组件的存在并不依赖于父容器，其父容器可以随时改变。

通常来说，AWT GUI 都是不可访问的。系统并没有为 AWT 程序员提供 API 来指定可访问性信息。可访问性（accessibility）处理的是残疾人可以怎样使用应用程序的问题。一个应用程序要想有很好的可访问性，必须与运行平台一起，让残疾人可以通过使用适当的辅助技术（提供其他用户接口的工具）来使用这些应用程序。很多政府和企业都有一些强制要求应用程序为实现可访问性而采用的标准。

Sun 希望 Java 语言能够成为一种“编写一次就可以随处运行（write once, run everywhere, 即 WORE）”的环境。这意味着可以在一台机器上开发和测试 Java 代码（例如在 Windows® 上），然后不经测试就可以在另外一个 Java 主机上运行同样的 Java 代码。对于大部分情况来说，Java 技术都可以成功实现这种功能，但是 AWT 却是一个弱点。由于 AWT 要依赖于主机 GUI 的对等体（peer）控件（其中每个 AWT 组件都有一个并行的主机控件或者对等体）来实现这个 GUI，这个 GUI 的外观和行为（这一点更重要）在不同的主机上会有所不同。这会导致出现“编写一次随处测试（write once, test everywhere, 即 WOTE）”的情况，这样就远远不能满足我们的要求了。

AWT 提供了一个丰富的图形环境，尤其是在 Java V1.2 及其以后版本中更是如此。通过 Graphics2D 对象和 Java2D、Java3D 服务，我们可以创建很多功能强大的图形应用程序，例如画图和制表包；结合使用 JavaSound，我们还可以创建非常有竞争力的交互式游戏。

9.2. Swing 概述

Java Swing 是 Java Foundation Classes (JFC) 的一部分，它是试图解决 AWT 缺点的一个尝试。在 Swing 中，Sun 开发了一个经过仔细设计的、灵活而强大的 GUI 工具包。不幸的是，这意味着我们又要花一些时间来学习 Swing 了，对于常见的情况来说，Swing 有些太复杂了。

Swing 是在 AWT 组件基础上构建的。所有 Swing 组件实际上也是 AWT 的一部分。Swing 使用了 AWT 的事件模型和支持类，例如 Colors、Images 和 Graphics。Swing 组件、布局管理器以及事件总结如下（参见 清单 4、清单 5 和 清单 6）。正如您可以看到的一样，这些组件集比 AWT 提供的组件集更为广泛，与 SWT 组件集相比也毫不逊色。

清单 4. 基本的 Swing Class 树（全部在 javax.swing 包或其子包中，“*” 表示抽象类）

Object	*Component	JMenuBar	JOptionPane
Container	*JComponent	JPanel	JPopupMenu
*AbstractButton	JButton	JProgressBar	JRootPane
JMenuItem	JCheckBoxMenuItem	JScrollBar	JScrollPane
JMenu	JRadioButtonMenuItem	JSeparator	JSlider
*JToggleButton	JCheckBox	JSplitPane	JTabbedPane
JRadioButton	Box	JComboBox	JDesktopIcon
Filler	JColorChooser	JFileChooser	JInternalFrame
JLabel	JLayeredPane	JDesktopPane	JList
JTable	JTableHeader	*JTextComponent	JEditorPane
FrameEditorPane	JTextPane	JTextArea	JTextField
JPasswordField	JToolBar	JToolTip	JTree
JViewport	ScrollableTab	Viewport Panel	Applet
JApplet	Window	Dialog	JDialog
Frame	JFrame	JWindow	

注意：在另外几个包中还有其他一些 Swing 组件，但是这是基本的组件集。

清单 5. Swing 提供了以下 LayoutManagers（全部在 javax.swing 包或其子包中，“*” 表示接口）

*LayoutManager CenterLayout *LayoutManager2 BoxLayout OverlayLayout SpringLayout

注意：在另外几个包中还有其他一些 Swing 布局管理器，很多都是为它们所布局的容器而专门定制的，但是这是基本的布局管理器集。

清单 6. Swing 提供了以下事件（大部分在 javax.swing.events 包及其子包中）

Object	EventObject	AWTEvent	AncestorEvent
ComponentEvent	InputEvent	KeyEvent	MenuKeyEvent
MouseEvent	MenuDragMouseEvent	InternalFrameEvent	

注意：在另外几个包中还有其他一些 AWT 事件，但是这是基本的事件集。这些是从更通

用的事件生成的“高级”事件。

为了克服在不同主机上行为也会不同的缺点，Swing 将对主机控件的依赖性降至了最低。实际上，Swing 只为诸如窗口和框架之类的顶层 组件使用对等体。大部分组件 (JComponent 及其子类) 都是使用纯 Java 代码来模拟的。这意味着 Swing 天生就可以在所有主机之间很好地进行移植。因此，Swing 通常看起来并不像是本地程序。实际上，它有很多外观，有些模拟 (尽管通常并不精确) 不同主机的外观，有些则提供了独特的外观。

Swing 对基于对等体的组件使用的术语是重量级 (heavyweight)，对于模拟的组件使用的术语是轻量级 (lightweight)。实际上，Swing 可以支持在一个 GUI 中混合使用重量级组件和轻量级组件，例如在一个 JContainer 中混合使用 AWT 和 Swing 控件，但是如果组件产生了重叠，就必须注意绘制这些控件的顺序。

Swing 无法充分利用硬件 GUI 加速器和专用主机 GUI 操作的优点。结果是 Swing 应用程序可能比本地 GUI 的程序运行速度都慢。Sun 花费了大量的精力来改进最近版本的 Swing (Java V1.4 和 1.5) 的性能，这种缺点正在变得日益微弱。由于 Swing 的设计更加健壮，因此其代码基础也更坚实。这意味着它可以在一台健壮的机器上比 AWT 和 SWT 上运行得更好。

除了具有更多的组件、布局管理器和事件之外，Swing 还有很多特性使得自己比 AWT 的功能更加强大。下面是更为重要的几个特性：

模型与视图和控件分离

对于这个模型中的所有组件 (例如按钮、列表、表、树、富文本) 来说，模型都是与组件分离的。这样可以根据应用程序的需求来采用模型，并在多个视图之间进行共享。为了方便起见，每个组件类型都提供有默认模型。

可编程外观

每个组件的外观 (外表以及如何处理输入事件) 都是由一个单独的、可动态替换的实现来进行控制的。这样我们就可以改变基于 Swing 的 GUI 的部分或全部外观。

呈现器和编辑器

大部分显示模型内容的组件，例如列表、表和树，都可以处理几乎所有类型的模型元素。这可以通过为每种组件类型和模型类型映射一个渲染器或编辑器来实现。例如，一个具有包含 java.util.Date 值的列的表可以有一些专用的代码来呈现数据值和编辑数据值。每一列都可以有不同的类型。

可访问性

创建一个残疾人可以访问的 GUI 是非常重要的。Swing 为实现具有可访问性的 GUI 提供了丰富的基础设施和 API。这种支持是单独的，但是如果主机上具有可访问性支持，那么

它们应该集成在一起。

与 AWT 一样，Swing 可以支持 GUI 组件的自动销毁。Swing 还可以支持 AWT 的自底向上和自顶向下的构建方法。

与 AWT 不同，Swing 组件不是线程安全的，这意味着您需要关心在应用程序中是哪个线程在更新 GUI。如果在使用线程时出现了错误，就可能会出现不可预测的行为，包括用户界面故障。有一些工具可以帮助管理线程的问题。

与 AWT 类似，Swing 的一个优点是，它也是 Java 技术的一种标准配置。这意味着您不需要自己来安装它了。不幸的是，Swing 已经有了很大的变化，因此它很容易变得依赖于最新版本的 Java 语言所提供的特性，这可能会强制用户更新自己的 Java 运行时环境。

9.3. SWT 概述

与 AWT 的概念相比，SWT 是一个低级的 GUI 工具包。JFace 是一组用来简化使用 SWT 构建 GUI 的增强组件和工具服务。SWT 的构建者从 AWT 和 Swing 实现中学习了很多经验，他们试图构建一个集二者优点于一体而没有二者的缺点的系统。从很多方面来说，他们已经成功了。

SWT 也是基于一个对等体实现的，在这一点上它与 AWT 非常类似。它克服了 AWT 所面临的 LCD 的问题，方法如下：定义了一组控件，它们可以用来构建大部分办公应用程序或开发者工具，然后可以按照逐个主机的原则，为特定主机所没有提供的控件创建模拟控件（这与 Swing 类似）。对于大部分现代主机来说，几乎所有的控件都是基于本地对等体的。这意味着基于 SWT 的 GUI 既具有主机外观，又具有主机的性能。这样就避免了使用 AWT 和 Swing 而引起的大部分问题。特定的主机具有一些低级功能控件，因此 SWT 提供了扩充（通常是模拟的）版本（通常使用 “C” 作为名字中的第一个字母），从而可以产生更一致的行为。

在对等体工作方式上，SWT 与 AWT 不同。在 SWT 中，对等体只是主机控件上的一些封装程序而已。在 AWT 中，对等体可以提供服务来最小化主机之间的差异（就是在这里，AWT 碰到了很多行为不一致的问题）。这意味着 SWT 应用程序实际上就是一个主机应用程序，它必然会全部继承主机的优点和缺点。这还意味着 SWT 不能完全实现 WORE 的目标；它更像是一种 WOTE 解决方案。这就是说，SWT 尽管不如 Swing 那么优秀，但是它在创建可移植解决方案方面是很杰出的。

SWT 部件、布局和事件总结如下（参见 清单 7、清单 8 和 清单 9）。正如您可以看到的一样，这些组件集比 AWT 提供的组件集更为广泛，与 Swing 组件集相比也毫不逊色。

清单 7. 基本的 SWT Class 树（大部分在 org.ecipse.swt.widgets 或 org.eclipse.swt.custom 包或子包中，“*” 表示抽象类，“!” 表示在 custom 包中，“~” 表示在其他包中）

Object	*Dialog	ColorDialog	DirectoryDialog
--------	---------	-------------	-----------------

FileDialog	FontDialog	MessageDialog	PrintDialog
*Widget	Menu	*Item	CoolItem
!CTabItem	MenuItem	TabItem	TableColumn
TableItem	TableTreeItem	ToolItem	TrayItem
TreeColumn	TreeItem	*Control	Button
Label	ProgressBar	Sash	Scale
Scrollable	Composite	~Browser	Canvas
*~AbstractHyperlink	~Hyperlink	~ImageHyperlink	*~ToggleHyperline
~TreeNode	~Twistie	AnimatedProgress	!CLabel
Decorations	Shell	FormText	StyledText
TableCursor	!CBanner	!CCombo	Combo
CoolBar	!CTabFolder	~ExpandableComposite	~Section
~FilteredList	~FilteredTree	~Form	Group
~PageBook	ProgressIndicator	!SashForm	!ScrolledComposite
TabFolder	Table	TableTree	ToolBar
Tray	Tree	ViewForm	Text
List	Slider		

注意：在另外几个包中还有其他一些 SWT 部件，但是这是基本的部件集。

与 AWT 和 Swing 布局管理器类似，SWT 也提供了非常丰富的布局部件集。布局系统与嵌套容器一起使用，可以生成所需要的任何布局算法。所有这 3 个 GUI 库也可以支持对部件的定位实现绝对控制。SWT 没有等效的 BorderLayout 部件，这一点非常令人失望。FormLayout 对于创建表单基本输入来说非常好用。我认为 SWT 的布局机制比 AWT/Swing 布局部件集的使用更难学习。

SWT 提供了以下布局管理器（大部分在 org.eclipse.swt.layout 和 org.eclipse.swt.custom 包或子包中，“*”表示接口，“!”表示在 custom 包中）

*Layout FillLayout FormLayout GridLayout RowLayout !StackLayout

注意：在另外几个包中还有其他一些 SWT 布局管理器，很多都是为它们所布局的容器而专门定制的，但是这是基本的布局管理器集。

与 AWT 和 Swing 事件系统一样，SWT 提供了非常丰富的事件集。尽管这些事件并不能与 AWT/Swing 的事件一一对应（例如 AWT 和 Swing 的按钮都会产生 ActionEvent 事件，而 SWT 的按钮产生的则是 SelectionEvent 事件），但是它们通常都是等价的。

清单 9. SWT 提供了以下事件（大部分在 org.eclipse.swt.events 包或 org.eclipse.swt.custom 包或其子包中，“*”表示抽象，“!”表示在 custom 包中）

Object	EventObject	SWTEventObject	TypedEvent
AimEvent	!BidiSegmentEvent	ControlEvent	!CTabFolderEvent
DisposeEvent	DragSourceEvent	DragTargetEvent	!ExtendedModifyEvent
focusEvent	HelpEvent	KeyEvent	TraverseEvent

VerifyEvent	!LineBackgroundEvent	!LineStyleEvent	MenuEvent
ModifyEvent	MouseEvent	PaintEvent	SelectionEvent
TreeEvent	ShellEvent	!TextChangedEvent	!TextChangingEvent

注意：在另外几个包中还有其他一些 SWT 事件，但是这是基本的事件集。这些是从更通用的事件生成的具体事件。

很多 Swing 组件，例如 JTable，都有自己的模型。对应的 SWT 控件（例如 Table）则没有；不过它们有自己的条目。条目通常用来限制显示文本或通常很小的图像（例如图标）。为了提供一种 Swing 的模型接口，SWT 使用了 JFace ContentProviders。这些组件可以在应用程序提供的模型（例如 List 或 Table 使用的 java.util.Array）和用作视图的控件之间充当一个桥梁。为了将任意模型对象格式化条目，SWT 使用了 JFace LabelProviders，它们可以为任何模型对象生成一个文本或图标格式。这可以对复杂模型对象的混合显示进行限制。其他类似组件，例如 ColorProviders 和 LabelDecorators，可以增强对这些条目的显示。对于 Tables 的特例来说，SWT 提供了 CellEditor，它可以临时将任意 SWT 控件链接到一个 Table 单元格上，从而当作这个单元格的编辑器使用。

SWT 不支持 GUI 控件的自动销毁。这意味着我们必须显式地销毁所创建的任何控件和资源，例如颜色和字体，而不能利用 API 调用来实现这种功能。这种工作从某种程度上来说得到了简化，因为容器控制了其子控件的自动销毁功能。

使用 SWT 只能自顶向下地构建 GUI。因此，如果没有父容器，子控件也就不存在了；通常父容器都不能在以后任意改变。这种方法不如 AWT/Swing 灵活。控件是在创建时被添加到父容器中的，在销毁时被从父容器中删除的。而且 SWT 对于 style 位的使用只会在构建时进行，这限制了有些 GUI 控件的灵活性。有些风格只是一些提示性的，它们在所有平台上的行为可能并不完全相同。

与 Swing 类似，SWT 组件也不是线程安全的，这意味着您必须要关心在应用程序中是哪个线程对 GUI 进行了更新。如果在使用线程时发生了错误，就会抛出异常。我认为这比不确定的 Swing 方法要好。有一些工具可以帮助管理线程的问题。

如果所支持的操作系统提供了可访问性服务，那么 SWT GUI 通常也就具有很好的可访问性。当默认信息不够时，SWT 为程序员提供了一个基本的 API 来指定可访问性信息。

SWT 提供了一个有限的图形环境。到目前为止，它对于 Java2D 和 Java3D 的支持还不怎么好。Eclipse 使用一个名为 Draw2D 的组件提供了另外一种单独的图形编辑框架（Graphical Editing Framework，GEF），它可以用来创建一些绘图应用程序，例如 UML 建模工具。不幸的是，GEF 难以单独（即在整体 Eclipse 环境之外）使用。

与 AWT 和 Swing 不同，SWT 和 JFace 并不是 Java 技术的标准配置。它们必须单独进行安装，这可以当作是 Eclipse 安装的一部分，也可以当作是单独的库进行安装。Eclipse 小组已经使它的安装变得非常简单，并且 SWT 可以与 Eclipse 分开单独运行。所需要的 Java 档案文件（JAR）和动态链接库（DLL）以及 UNIX® 和 Macintosh 上使用的类似库可以从 Eclipse Web 站点上单独下载。JFace 库需要您下载所有的 Eclipse 文件，并拷贝所需要

的 JAR 文件。在下载所需要的文件之后，我们还需要将这些 JAR 文件放到 Java CLASSPATH 中，并将 DLL 文件放到系统 PATH 中。

特性的比较

表 1. SWT 、AWT 和 Swing 特性的比较

功能/角色/外表	AWT	Swing	SWT（风格）
显示静态文本	Label	JLabel	Label, CLabel
显示多行静态文本	Multiple Labels	具有 HTML 内容的 Multiple JLabels 或 JLabel	具有新行的 Multiple Labels 或 Label
显示多行格式化静态文本	具有不同字体的 Multiple Labels	具有 HTML 内容的 JLabel	具有不同字体的 Multiple Labels
单行文本输入	TextField	JTextField	Text(SWT.SINGLE)
多行文本输入	TextArea	JTextArea	Text(SWT.MULTI)
显示图像	N/A	JLabel	Label
显示文本和图像	N/A	JLabel	CLabel
提示弹出帮助	N/A	组件的 setToolTip , JToolTip 子类	控件的 setToolTip
风格化的文本输入	N/A	JEditorPane	StyledText
从条目列表中进行选择	List	JList	List
简单按下具有文本的按钮	Button	JButton	Button(SWT.PUSH)
简单按下具有文本或图像的按钮	N/A	JButton	Button(SWT.PUSH)
绘图区域；可能用于定制控件	Canvas	JPanel	Canvas
选中/取消复选框	CheckBox	JCheckBox	Button(SWT.CHECK)
单选按钮选择	CheckBoxGroup	ButtonGroup 和 Menu	Group 和 Menu
从一个下拉列表中选择	Choice	JComboBox	Combo、CCombo

输入文本或从下拉列表中选择	N/A	JComboBox	Combo、CCombo
可滚动区域	ScrollPane	JScrollPane	创建 Scrollable 子类
顶层窗口	Dialog、Frame、Window	JDialog、JFrame、JWindow	具有不同风格的 Shell
通用窗口	Window	JWindow	Shell
框架窗口	Frame	JFrame	Shell(SWT.SHELL_TRIM)
对话框窗口	Dialog	JDialog	Shell(SWT.DIALOG_TRIM)
菜单	Menu	JMenu	Menu
MenuItem	MenuItem	JMenuItem	MenuItem
菜单快捷键	通用击键	与 AWT 相同	依赖于主机的快捷键
弹出菜单	PopupMenu	JPopupMenu	Menu(SWT.POPUP)
菜单条	MenuBar	JMenuBar	Menu(SWT.BAR)
显示插入符号	N/A	Caret	Caret
Web 浏览器	N/A	JTextPane (HTML 3.2)	Browser (通过嵌入式浏览器)
Web 页面中的嵌入式控件	Applet	JApplet	主机控件 (例如 OLE)
其他控件的通用容器	Panel	JPanel	Composite
其他控件的有边界通用容器	Panel (如果是手工画的)	具有 Border 的 JPanel	Composite(SWT.BORDER)
其他控件的有边界和标题的通用容器	N/A	具有 TitledBorder 的 JPanel	Group
单选按钮 (一个被选中)	Checkbox	JRadioButton	Button(SWT.RADIO)
单选按钮的控件扩充	CheckboxGroup	RadioButtonsGroup	Group
箭头按钮	N/A	具有图像的 JButton	Button(SWT.ARROW)
支持文本显示方向	通过 ComponentOrientation	与 AWT 相同	很多组件都可以支持这种风格
焦点切换	Policy 和 Manager 对象	与 AWT 相同	下一个控件
定制对话框	Dialog 子类	JDialog 子类	Dialog 子类
访问系统事件	EventQueue 服务	与 AWT 相同	Display 服务 (不如 AWT 健壮)

系统访问对话框	FileDialog	JColorChooser 、 JFileChooser	ColorDialog 、 DirectoryDialog、 FileDialog、 FontDialog、 PrintDialog
显示简单消息对话框	N/A（必须是 Dialog 子类）	JOptionPane 静态方法	具有很多风格的 MessageBox
显示简单提示对话框	N/A（必须是 Dialog 子类）	JOptionPane 静态方法	N/A（JFace 中用来实现这种功能的子类）
布局管理器	BorderLayout 、 CardLayout、 FlowLayout、 GridLayout 、 GridBagLayout	AWT 加上 BoxLayout 、 CenterLayout 、 SpringLayout	FillLayout 、 FormLayout 、 GridLayout 、 RowLayout 、 StackLayout
基本的绘图控件	Canvas	JPanel	Canvas
基本绘图	Graphics 和 Graphics2D 对象 —— 基本形状和文本，任意 Shapes 和 Strokes、Bezier 以及文件	与 AWT 相同	GC 对象 —— 基本形状和文本
绘图转换	Affine，合成	与 AWT 相同	N/A
离屏绘图（Off screen drawing）	BufferedImage 、 drawImage	与 AWT 相同	Image、drawImage
双缓冲区	手工	自动或手工	除非由主机控件提供，否则就是手工
打印	PrintJob 和 PrintGraphics	与 AWT 相同	向 Printer 设备绘图
定制颜色	Color	与 AWT 相同	Color
定制字体	Font、FontMetrics	与 AWT 相同	Font
光标选择	Cursor	与 AWT 相同	Cursor
图像特性	从文件中加载，动态创建，可扩充地编辑	与 AWT 相同	从文件中加载，动态创建，基本编辑
输入自动化	Robot	与 AWT 相同	N/A
显示工具条	N/A	JToolBar	ToolBar、CoolBar
显示进度条	N/A	JProgressBar	ProgressBar
将空间划分成区域	N/A	JSplitPane	Sash 或 SashForm
显示一个分标签页的区域	N/A	JTabbedPane	TabFolder、CTabFolder
显示制表信息	N/A	JTable	Table
格式化表的列	N/A	TableColumn	TableColumn
显示层次化	N/A	JTree	Tree

信息			
从一定范围的值中进行选择	N/A	JSlider	Slider
从一组离散范围的值中进行选择	N/A	JSpinner	Scale
对于基本显示的访问	Toolkit GraphicsConfiguration GraphicsDevice	与 AWT 相同	Display
将条目添加到系统托盘（system tray）中	N/A	N/A	Tray
关键： N/A —— 不适用。在很多情况中，这种特性都可以通过创建定制控件或控件容器或利用其他定制编程来实现，不过实现的难度会有所不同。			

本章对 Eclipse 的 Standard Windows Toolkit with JFace、Java 的 Swing 和 Abstract Windows Toolkit GUI 工具包进行了比较。通过此处提供的比较，您可以确定在自己的新应用程序中应该使用哪个 GUI 工具包。

10 结束语

通常来说，每个工具包都非常完整且功能强大，足以构建功能完善的 GUI，但是 Swing 通常要比单独使用 SWT（不使用 JFace 时）更好。Swing 具有内嵌于 Java 技术的优点，是完全可移植的，无可争议地是一种更好的架构。Swing 也具有高级图形应用程序所需要的优点。SWT 具有可以作为本地应用程序实现的优点，这可以提高性能，并利用基于 SWT 的 GUI 来实现本地兼容性。

如果您只为一种平台来开发系统，那么 SWT 就具有主机兼容性方面的优点，包括与主机特性的集成，例如在 Windows 上对 ActiveX 控件的使用。

同时也应该看到，无论任何一个组件，工具包，甚至扩展到一种编程语言，建立的体系都是相当完整的，同时也是在不断的完善，发展进步的，我们要学习和感悟的是组件封装的体系，，只有理解好组件设计思想，工作中才能灵活的运用。设计优良的程序仰赖与日积月累的程序设计经验，和逻辑业务理解能力，希望更多的人加入到这个群体中共同努力。