SWT/Jface 开发入门指南

写在前面的话

终于决定提起笔来写一篇关于 swt 和 JFace 编程的文章。在开始之前，我想先介绍一下你能够从将要出现的这一系列文章里得到什么，以及更重要的，**你不能得到什么**。我们的时间是如此之重要，以至于我们很难容忍把它浪费在自己不关心的事情上。

因为我刚开始写，所以到底这些文章会分成几次发布出来，也很难讲。但是我心里大体有这样一个提纲。也就是说，我打算介绍以下方面的内容：

- 设定 swt 以及 JFace 的开发环境
- swt 的一些简单部件（widget）介绍
- JFace 的一些入门性介绍
- swt 和 JFace 的事件模式
- 通过一些简单的例子说明如何利用 swt 和 JFace 编写图形化应用程序

我还要说明一下你**不能**从本文中得到的信息，这些信息包括：

- swt 和 JFace 相关内容非常深入的介绍：正如你从题目中了解到的，本文的定位是“入门”，也就是说假定的读者是那些对 swt 和 JFace 开发没有什么了解的人。所以我并不打算进行一个深入的介绍。因为我相信在入门之后，他们

能够找到更好的资料（此外，我可能会写另外一个系列文章来介绍）。

- Eclipse 的使用：我假定你在读这篇文章的时候已经对 Eclipse 有所了解，所以不会解释到具体 Eclipse 如何使用。
- Eclipse 插件开发：虽然 Eclipse 插件开发和本文内容有着千丝万缕的联系，我决定还是不把它列为介绍的内容。这方面，你仍然可以找到相当多的资料供参考。

与作者联系

如果你希望和我联系的话，你可以发email到jayliu@mail.csdn.net。

我的blog：http://www.blogjava.net/jayliu/

延伸阅读

你不知道什么是eclipse么？你可以打开http://www.eclipse.org/，这是eclipse的官方站点。

如果你希望对eclipse功能有详细的了解，你可以读一下这本书《CONTRIBUTING TO ECLIPSE》。

你不知道什么是swt/JFace么？你可以看一下Wikipedia中的介绍：

http://eclipsewiki.editme.com/SWT

http://en.wikipedia.org/wiki/Eclipse_%28computing%29

你对eclipse,swt的历史感兴趣么？我强烈推荐你看一下这篇八卦：

http://www.csdn.net/news/newstopic/20/20433.shtml

如果你对eclipse的一些新闻比较感兴趣的话，可以关注一下大胃的blog:http://www.blogjava.net/sean/以及kukoo的blog:http://www.blogjava.net/kukoo/
最后你可以从一个地方得到所有你想要的东西：http://www.google.com/

OK，介绍到此为止，下面我开始正文：

Hello,world!:搭建一个 swt/JFace 开发环境

在前面我曾经提到过：我们假定你对 Eclipse 开发有一些了解。所以在这一节中，我将示范如何搭建一个 swt/JFace 开发环境，并且用一个老掉牙的 Hello,world!程序作为示范。

建立开发环境

你完全可以不使用 Eclipse，而是使用别的 IDE 来进行开发，但是你需要有 swt/JFace 的一些库文件。

为了能够进行正常的 swt/JFace 开发，你需要做以下工作：

第一步：建立一个 Java 工程：因为这个是大家都非常熟悉的，所以就不再赘述，你可以建立一个任何名字的 Java 工程。

第二步：导入 swt 以及 JFace 的库文件。这也就意味着：将 swt/JFace 相关的库文件导入到工程的 classpath 中去。

需要的库文件有哪些呢？打开 eclipse 安装目录下的 plugins 文件夹，我们需要找到以下 jar 文件：

- org.eclipse.swt_3.x.x.jar
- org.eclipse.jface_3.x.x.jar

- org.eclipse.ui.workbench_3.x.x.jar

这就是我们搭建一个基本 swt/eclipse 程序所需要基本的一些库文件了。其中 3.x.x 视你所使用的 eclipse 版本而定，譬如我的 eclipse 版本是 3.1M6，这些 3.x.x 就是 3.1.0。将他们加入你程序的 classpath 中。

为了更加清楚地说明如何做，你可以看一下下面的图 1。这是我在 eclipse 的做法：打开工程的 properties 对话框，然后选择 Java Build Path 中的 Libraries 选项卡，将这些 jar 导入进来。当然你可以有自己的做法。

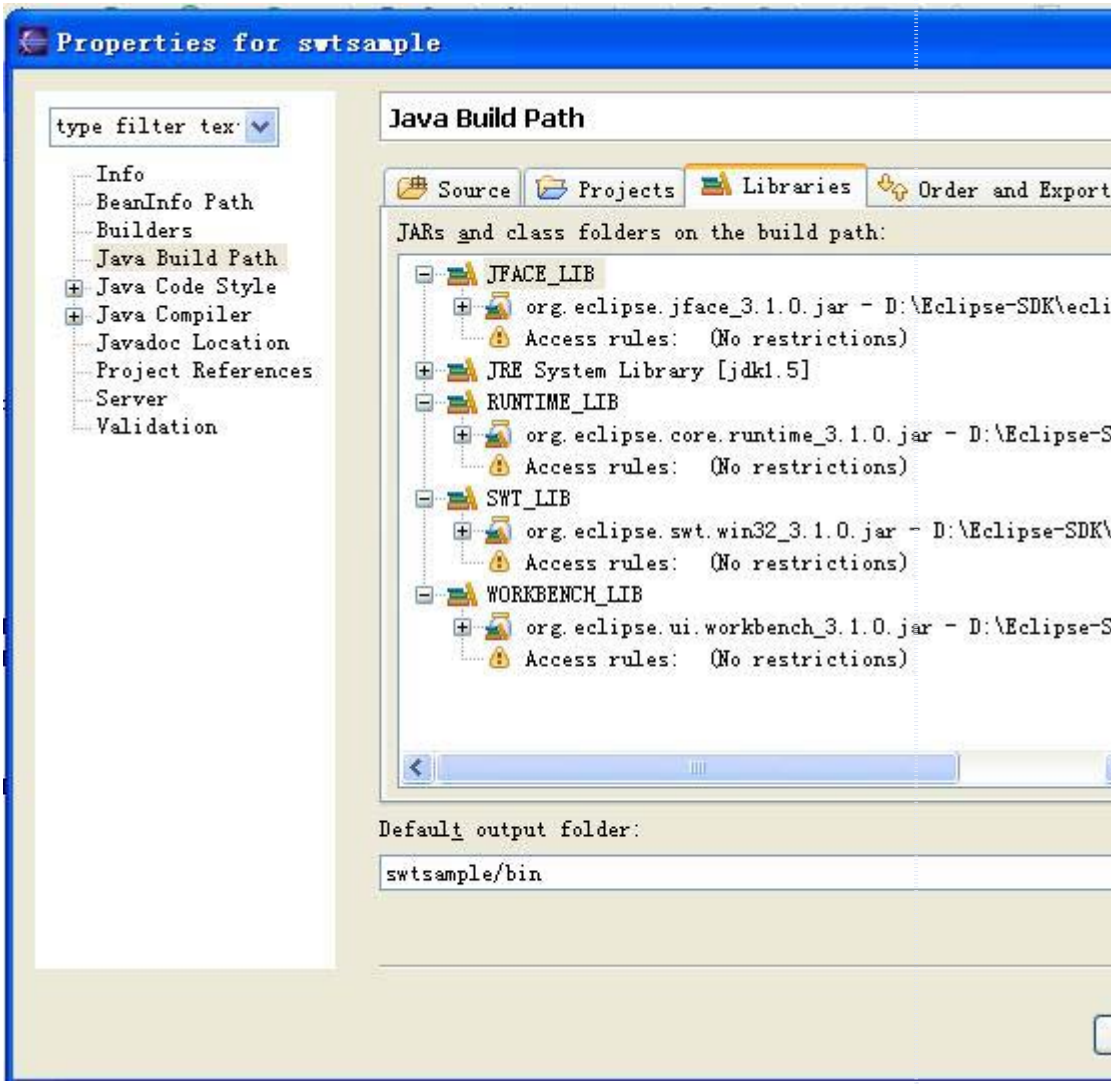

图 1

第三步：为你的 java 程序添加本地库文件。如果你使用 windows 的话，你可能注意到在 eclipse 的 plugins 目录下还有一个 org.eclipse.swt.win32_3.x.x.jar，将这个 jar 解压以后在 os/win32/x86 目录下有几个 dll 文件。这几个 dll 为 swt 通过 JNI 访问 windows 本地 API 提供了接口，我们需要将使 java 程序在启动时候即能够访问它。你可以有多种办法实现这个目的：

最简单的办法就是直接把这几个文件拷贝到你 jdk 的 bin 目录下

你也可以设定环境变量时候，在 PATH 中加入这几个 dll 文件的目录。

你还可以在运行程序时候指定 java.library.path 为这几个 dll 所在的目录，在 eclipse 中，如果你打开 Help 菜单中 About Eclipse Platform，然后在出现的对话框中选择 configuration details 按钮，你可以在接下来出现的对话框中找到 java.library.path

配置 java.library.path 你还有另外一种办法，就是在最后运行程序的时候，从 Run 菜单中选择 Run As..而不是 Run As Java Application，填写好必要的其他参数以后打开 Argument 选项卡，然后在 VM Argument 输入框中填写该参数为 dll 文件的路径，如图 2 所示

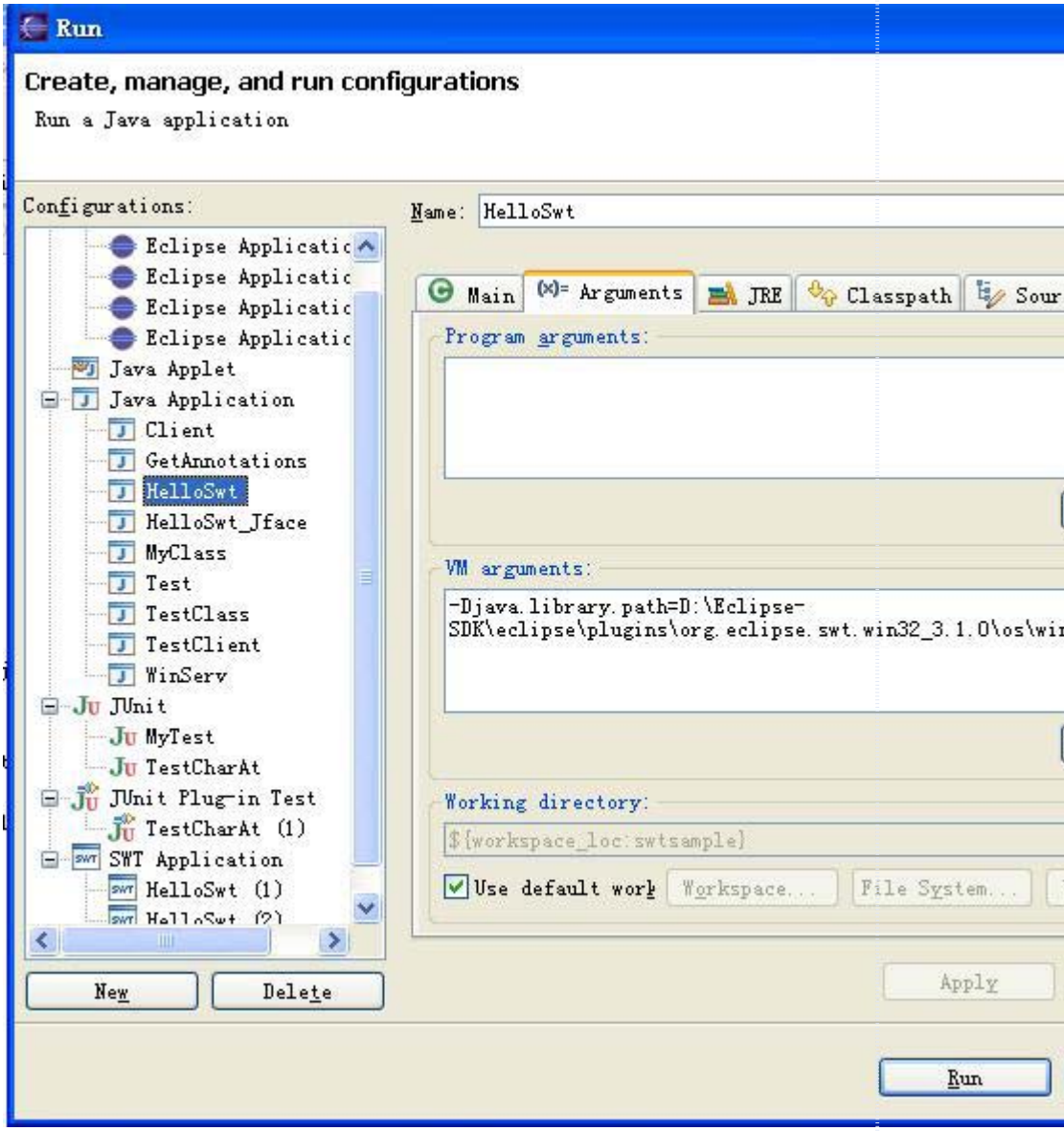

图 2

通过这三步工作，你就可以编写 swt/JFace 程序了。

Hello,World!

下面的内容就是我们 `Hello,world!` 的示例。首先建立一个类，我将这个类取名为 `HelloSwt`，在我的工程中，它位于 `swtjfacesample` 包的下面。类的内容如下：

```
1  package swtjfacesample;
2
3  import org.eclipse.swt.SWT;
4  import org.eclipse.swt.widgets.Display;
5  import org.eclipse.swt.widgets.Shell;
6  import org.eclipse.swt.widgets.Text;
7
8  public class HelloSwt {
9      /**
10     * Hello,world!
11     *
12     * @param args
13     */
14     public static void main(String[] args) {
15         Display display = new Display();
16         Shell shell = new Shell(display);
17
18         Text helloText = new Text(shell, SWT.CENTER);
19         helloText.setText("Hello,World!");
20         helloText.pack();
21
22         shell.pack();
23         shell.open();
```

```
24
25          while (!shell.isDisposed()) {
26              if (!display.readAndDispatch()) {
27                  display.sleep();
28              }
29          }
30          display.dispose();
31
32      }
33  }
34
```

代码段 1

关于这段代码的内容，我们会在下面的内容中进行详细介绍。现在我们可以尝试着运行一下，确定已经编译完成后从eclipse的Package Explorer中选中这个类然后点右键，在弹出的菜单中你会看到Run As，进一步选中这一项，然后在二级菜单中选“Run As Java Application”，如果运行正常的话你会看到如图 3 的运行结果：

图 3

Q&A:出现了问题怎么办

Q:我的程序编译时候出现了错误!

A:如果是提示诸如 Text,Display 这些类不能够被成功引入，那么你需要检查一下是否我们前面提到的 jar 都已经被成功引入到你的 classpath

Q:编译成功了但是不能运行，出现了异常

A:检查一下异常信息，如果你的异常信息类似这样：

```
Exception in thread "main" java.lang.UnsatisfiedLinkError: no
swt-win32-3128 in java.library.path

...
```

的话，那说明我们在前面第三步提到的那些 dll 没有正常工作，最好能够按照前面的说明重新配置一下。

SWT/Jface开发入门指南（二）

从 Hello, world!开始了解 Display 和 Shell

在这一节中，我们从前面所列举出来的 `Hello, world!`程序开始对 `swt` 进行一些初步的探索。所谓的初步是指，我们会介绍编写 `swt` 程序的基本思路，以及对两个重要的类:`Display` 和 `Shell` 作一些介绍。

因为这一节和前一节是分成两个部分贴出来的，所以我仍然将 `Hello, world!`的代码段在下面列出来：

```
1   package swtjfacesample;
2
3   import org.eclipse.swt.SWT;
4   import org.eclipse.swt.widgets.Display;
5   import org.eclipse.swt.widgets.Shell;
6   import org.eclipse.swt.widgets.Text;
7
8       public class HelloSwt {
9           /**
10      * Hello,world!
11      *
12      * @param args
13      */
14              public static void main(String[] args) {
15          Display display = new Display();
16          Shell shell = new Shell(display);
17
```

```
18              Text helloText = new Text(shell, SWT.CENTER);
19              helloText.setText("Hello,World!");
20              helloText.pack();
21
22              shell.pack();
23              shell.open();
24
25                  while (!shell.isDisposed()) {
26                      if (!display.readAndDispatch()) {
27                    display.sleep();
28                  }
29              }
30              display.dispose();
31
32          }
33      }
```

代码段 2

这段程序虽然很简单，但是它反映了我们书写 swt 程序的步骤，这些步骤是：

1. 创建一个 Display 对象
2. 创建一个或者多个 Shell 对象，你可以认为 Shell 代表了程序的窗口。
3. 在 Shell 内创建各种部件（widget）
4. 对各个部件进行初始化（外观，状态等），同时为各种部件的事件创建监听器（listener）
5. 调用 Shell 对象的 open()方法以显示窗体
6. 各种事件进行监听并处理，直到程序发出退出消息

7. 调用 Display 对象的 dispose()方法以结束程序。

在 Hello,world!程序中，为了让程序更加简单，我们没有创建事件监听器，在以后的内容中会进行专门介绍。

现在让我们稍微深入一些，看一下这些 Display,Shell 有什么作用以至于我们每个程序都必须有它们存在。

Display:与操作系统沟通的桥梁

我们在前面说过,每个 swt 程序在最开始都必须创建一个 Display 对象。Display 对象起什么作用呢？它是 swt 与操作系统沟通的一座桥梁。它负责 swt 和操作系统之间的通信。它将 swt/JFace 的各种调用转化为系统的底层调用，控制操作系统为 swt 分配的资源。同时我们也可以通过 Display 对象得到操作系统的一些信息。

Display 是一个“幕后工作者”，它为 swt/JFace 提供支持，但是你并不能够从某个用户界面中看到它的影子。

在前面的 Hello,world!程序中，我们可以看到构建一个 Display 对象是和普通的 Java 对象一样通过构造函数实现的。它为实现图形界面准备了最基本的条件。而在程序结束时我们必须显式地调用 dispose() 方法来释放程序运行中所获得的资源。一般来说，一个程序只需要一个 Display 对象，当然没有人禁止你创建多个 Display 对象。但是在 swt 的 javadoc 中，我们可以看到关于这个问题一些描述：

“Applications which are built with SWT will *almost always* require only a single display. In particular, some platforms which SWT supports will not allow more than one *active* display. In other words, some platforms do not support creating a new display if one already exists that has not been sent the dispose() message.”

Display 有着众多的方法，我们不可能一一介绍。在这里只挑选几个可能会比较常用的作一些简单介绍。

- setData()和 getData()：这一对函数允许我们为 Display 对象设定一些数据,setData()的参数中 key 和 value 类似于我们在使用 Map 对象中 key 和 value 的含义。
- getShells()得到关联到该 Display 对象的所有没有 dispose 的 Shell 对象
- getCurrent()得到与用户交互的当前线程
- readAndDispatch()得到事件并且调用对应的监听器进行处理
- sleep()等待事件发生

Shell:窗口

一个 Shell 对象就是一个窗口。你可以在上面放置各种部件创建丰富的图形界面。

我们都知道窗口有很多种，比如窗口有可以调整大小的，有不可以的，有的没有最大化最小化按钮。这些窗体的特征在 swt 中被成为风格（style）。一个窗体的风格可以用一个整数进行定义。这些风格的定义在 org.eclipse.swt.SWT 中。

Shell 对象可用的风格包括：BORDER, CLOSE, MIN, MAX, NO_TRIM, RESIZE, TITLE , PLICATION_MODAL, MODELESS, PRIMARY_MODAL,S YSTEM_MODAL

这些风格我们不作一一介绍，你可以从他们字面意义看出一些含义来，当然也可以参考对应的 javadoc。

我们可以在一个 Shell 的构造函数中定义它的风格，比如在前面的 Hello,world! 程序中，我们可以这样定义 Shell。

```
Shell shell = new Shell(display,SWT.CLOSE | SWT.SYSTEM_MODAL);
```

最后得到的窗体没有最大化和最小化按钮，并且大小是固定不变的。

因为 swt 运行于各种平台之上，而这些平台上的窗口管理器千差万别，所以所有这些风格都不是肯定可以实现的。在 swt 的 javadoc 中，这被称为暗示（hints）。

Shell 对象的方法大都和 GUI 有关，比如 setEnabled()设定了窗体是否能够和用户进行交互，setVisble()设定了窗体是否可见，setActive()将窗体设为当前的活动窗口。

我们可以用 open()方法打开一个窗体，close()方法关闭一个窗体。

小结

本节讨论了 Display 和 Shell 的一些概念，这是我们以后进一步了解 swt 的基础。在下一节中，我将介绍各种部件（widget）的用法，所谓部件，是指文本框，标签等 UI 实体。

SWT/Jface开发入门指南（三）

初步体验 widget

大家好，五一已经过去了，你们玩得开心么？

在前面的两篇文章中，我向大家介绍了 swt 的一些基本知识，现在让我们继续下去讨论一下 swt 中的 widget 相关的一些知识以及介绍几种最为简单的 widget。

从 Widget 和 Control 开始

Widget 是 Control 的父类，而 Control 是我们使用的大多数部件的父类。我们在以前的一些编程语言中可能接触过“控件”或者“组件”之类的概念，部件（widget）的概念大体和这些相当。

在 org.eclipse.swt.widgets 中定义了众多的 widget，甚至我们前面介绍的 Shell 也被当成 widget 的一种。

因为可用的 widget 如此之多，所以我大概没有办法全部一一介绍。在这一节中，我会介绍几种常用的 widget。相信善于触类旁通的你通过这些极为简略的介绍应该可以开始使用各种 widget，在使用中不断完善自己的认识。

首先我们来介绍 Widget。它是一个**抽象类**，也是所有 widget 的父类。通过介绍这个类，我们可以得出这所有 widget 的一些共有特性。

Widget 的方法中 dispose()方法我们在以前曾经见到过，调用这个方法时候，所谓的接收者（receiver，譬如我们这样写：awidget.dispose()，那么 awidget 就是接收者，而这句话所处的对象成为调用者或者 caller）和接收者中所包含的其他 widget 会释放它们所占用底层操作系统的资源。这也就是说你不必显式地为程序中创建的每个 widget 进行 dispose()调用，而只需要确保最外层的 widget（比如 Display）进行了 dispose()就可以了。

另外还可以通过 isDisposed()判断是否该 widget（接收者）已经进行了 dispose。
Widget 中的 getStyle()方法得到 widget 的风格，而 getDisplay()得到所处的 Display 对象。

此外 Widget 中还有两个重要方法 getData()和 setData()，这两个方法允许我们为一个 widget 附加其他的信息。特别是在你需要跨模块传递 widget 的时候，这个功能显得非常有用。比如如果一个文本框中显示了一段文章中的某句话，如果我们同时希望把这整篇文章的题目和作者附加上的话可以这样写：

```
1 atext.setData("title","I Have A Dream");
2 atext.setData("author","Martin Luther King");
```

代码段 3

在程序的其他部分可以用 atext.getData(“title”)得到这篇文章的题目，或者用 atext.getData("author")得到作者。

在前面我们提到过，Control 是今后我们所使用大部分 widget 的父类。在这里我不单独进行介绍，而是在后面的部分中介绍。

创建部件

和创建其他 java object 一样，我们通过使用 new 操作符创建部件的实例。有一点比较特殊的可能你需要使用带参数的构造函数进行 new 操作。类似下面的程序：

```
Text text=new Text(shell,SWT.CENTER);
```

这种方法适用于几乎所有的 widget，其中第一个参数是父 widget，也就是指明了该widget 需要被放置道另外哪一个widget 之中,而第二个参数是这个widget 的风格。

大小和位置

仅仅创建一个部件并不足以让你看到它，因为一个部件初始的长和宽都是 0。你还需要设定它的大小。你可以用 setSize()或者 setBounds()方法手动设定部件的大小，也可以让系统自动调整部件的大小到一个合适的值，这个值也被成为首选尺寸（preferred size）。

可以通过调用 pack()方法让系统调整控件大小。如果你希望系统自动调整，那么你需要首先设定控件需要表达的内容。举个例子来说，如果你的部件是一个文本框或者标签，你应该首先设定它所要显示的文本，这样系统可以通过文本的长度计算。

对于部件的位置，同样可以使用 setLocation()或者 setBounds()进行设定。

这里值得一提的是所谓的 bounds，其实 bounds 可以看成是大小和尺寸的综合。比如 setBounds(int x, int y,int width,int height)的参数中，x 和 y 描述的是位置信息，而 width 和 height 描述了大小。

隐藏与失效

通过部件的 setVisible 方法可以控制部件进行隐藏或是显示。通过 setEnabled 方法可以控制部件是否有效。一个无效的部件不会对用户的任何动作作出响应。这两个方法的参数都是布尔型的。

提示文本

可以通过 setToolTipText()方法设定部件的提示文本。

几种常用的部件

Label

标签用来显示静态的文本或者图像。关于图像和色彩我会在后面的部分进行介绍。

标签可以使用 SWT.CENTER, SWT.LEFT, SWT.RIGHT 中的一种指明文本的对齐方式（居中对齐，左对齐，右对齐）。

你也可以通过设置标签属性为 SWT. SEPARATOR 使标签成为一条分隔符。

下面这个程序给出了标签的两种表现形式，其中使用了 Layout，关于 Layout 的详细情况也会在后面的部分进行介绍：

```
1 public class Labels {
2
3     public static void main(String[] args) {
4         Display display = new Display();
5         Shell shell = new Shell(display,SWT.SHELL_TRIM);
6         RowLayout layout=new RowLayout(SWT.VERTICAL);
7         shell.setLayout(layout);
8         shell.setText("Labels");
9         Label label1=new Label(shell,SWT.CENTER);
10         label1.setText("Label Demo");
11         Label label2=new Label(shell,SWT.SEPARATOR | SWT.HORIZONTAL);
12         shell.setSize(100,100);
13         shell.open();
14
15         while (!shell.isDisposed()) {
16             if (!display.readAndDispatch()) {
17                 display.sleep();
18             }
```

```
19        }
20        display.dispose();
21    }
22}
23
```

代码段 4

最后得到的效果可以参照下图：

图 4

Text

Text 就是最简单的文本框，与标签一样，我们可以通过设定它的风格来表示它的对齐方式（SWT.CENTER，SWT.LEFT，SWT.RIGHT），另外还有其他一些用于文本支持的方法，比如 insert()，paster()，copy()，setSelection()，selectAll()等，这些方法在后面介绍 swt 事件模式会进行更详细的介绍。

Button

在 swt 中，Button 并不仅仅是按钮。构造时候定义的风格不同，所体现出的外观也不一样。

如果风格定义成 SWT.PUSH，它就是一个普通的按钮。

如果定义为 SWT.TOGGLE，它在被按下以后会保持按下的形状（而不会弹起来），直到鼠标再次在上面按一下才会回复弹起的形状。

如果风格定义为 SWT.ARROW，它是一个带箭头的按钮，箭头的指向可以选择 SWT.LEFT，SWT.RIGHT，SWT.UP，SWT.DOWN 中的一个。

如果定义为 SWT.CHECK，它是一个复选框。

如果定义为 SWT.RADIO，它是一个单选框。

下面一段程序演示了各种不同的 Button。

```
1 public class Buttons {
2
3         public static void main(String[] args) {
4             Display display = new Display();
5             Shell shell = new Shell(display,SWT.SHELL_TRIM);
6             RowLayout layout=new RowLayout(SWT.VERTICAL);
7             shell.setLayout(layout);
8             shell.setText("Buttons");
9
10            Button pushbutton=new Button(shell,SWT.PUSH | SWT.CEN
TER);
11            pushbutton.setText("SWT.PUSH");
12
13            Button togglebutton=new Button(shell,SWT.TOGGLE | SW
T.LEFT);
14            togglebutton.setText("SWT.TOGGLE");
```

```
15              togglebutton.setSelection(true);
16
17              Button arrowbutton=new Button(shell,SWT.ARROW | SWT.L
EFT);
18
19
20              Button checkbox=new Button(shell,SWT.CHECK);
21              checkbox.setText("SWT.CHECK");
22
23              Button radio=new Button(shell,SWT.RADIO);
24              radio.setText("SWT.RADIO");
25              radio.setSelection(true);
26
27              shell.pack();
28              shell.open();
29
30               while (!shell.isDisposed()) {
31                   if (!display.readAndDispatch()) {
32                      display.sleep();
33                  }
34              }
35              display.dispose();
36      }
37  }
```

代码段 5

最后得到的窗口如下图：

图 5

小结

在这一节中我向大家介绍了 widget 的一些基本知识，还有几种简单的 widget。你可能注意到这些描述仅仅限于外观方面，如何让 widget 和用户交互起来呢？这需要我们处理各种用户事件，在下一节中我会向大家介绍 swt 的事件模式。

SWT/JFace开发入门指南（四）

让你的 swt 程序动起来

在向使用者提供最差的用户体验方面，中国的 IT 企业始终走在时代的最前端。之所以有这样的感慨其实是来源于往 blog 上贴上一节的内容：我用了一整天的功夫，不断与 CSDN 各种莫名其妙的出错提示进行斗争，最后终于成功的贴了上去。

其实作为 CSDN blog 一个使用者，我的要求并不高：只要能写 blog，能够正常访问就可以了。然而就是这么一点基本的要求好像也得不到满足。

我不知道大家有没有这样的体验：其实软件使用者要求的东西都很基本，而现在软件做得越来越复杂，有相当大一部分是在于软件开发者把自己的注意力放在了一些附加功能（这些功能可能让用户感到惊喜，但是如果没有它们用户也不会不满意）上，而真正用户的要求却得不到满足。所以大家在设计程序的时候，一定要明白，有时候简单就是一种美，把时间花费到真正有价值的地方去。

OK，回到我们的主题上来。在这一节中，我将给大家介绍 swt 的事件模式。在前面我们也提到过，写一个 swt 程序，无非就是分几步走。其中比较需要费心的就是布置好用户界面和处理各种事件。

添加了事件处理的 Hello, world!

其实 swt 中处理事件非常简单，对应于各种事件都有相应的 listener 类，如果一种事件叫做 Xyz，那么对应的 listener 类就是 XyzListener。比如对应于鼠标事件的有 MouseListener，对应于键盘事件的就是 KeyListener。而在每种 widget 中，对于它可以处理的事件都有 addXyzListener 方法，只要把对应的 listener 实例作为参数传给它就可以了。

为了更加清楚的说明，我们先来看下面一段程序：

```
1 public class EventDemo {
2
3   private Shell _shell;
4
5   public EventDemo() {
6      Display display = new Display();
7      Shell shell = new Shell(display,SWT.SHELL_TRIM);
8      setShell(shell);
9      RowLayout layout=new RowLayout();
10      shell.setLayout(layout);
11      shell.setText("Event demo");
12
13      Button button=new Button(shell,SWT.PUSH | SWT.CENTER);
14      button.setText("Click me!");
15
16      button.addSelectionListener(new SelectionListener(){
```

```
17
18          public void widgetSelected(SelectionEvent event) {
19              handleSelectionEvent();
20          }
21
22          public void widgetDefaultSelected(SelectionEvent event) {
23          }
24      });
25      shell.setBounds(200,300,100,100);
26      shell.open();
27
28      while (!shell.isDisposed()) {
29          if (!display.readAndDispatch()) {
30              display.sleep();
31          }
32      }
33      display.dispose();
34
35  }
36
37  protected void handleSelectionEvent() {
38      MessageBox dialog=new MessageBox(getShell(),SWT.OK|SWT.ICON_INFORMATIO
N);
39      dialog.setText("Hello");
40      dialog.setMessage("Hello,world!");
41      dialog.open();
42  }
43
44  /**
45   * @param args
46   */
47  public static void main(String[] args) {
48
49      EventDemo eventdemo=new EventDemo();
50  }
51
52  /**
53   * @return Returns the _shell.
54   */
55  public Shell getShell() {
56      return _shell;
57  }
58
59  /**
```

```
60     * @param _shell The _shell to set.
61     */
62    public void setShell(Shell shell) {
63        this._shell =shell;
64    }
65 }
66
```

代码段 6

你可以看到在这段程序中，我们只创建了一个 Button，随后调用了它的 addSelectionListener()方法，在这个新创建的 Listener，我们只为 widgetSelected 方法添加了代码，并在其中创建了一个对话框。实际运行效果如下图，其中那个标有 Hello,world 的对话框是按了按钮以后出现的：

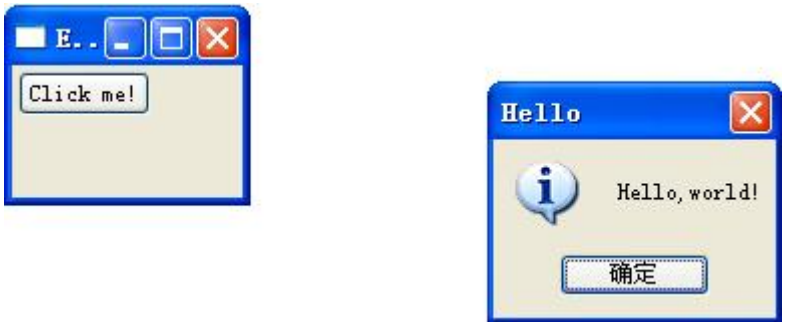

图 6

如果总结一下，我们可以得出处理事件的几个步骤：

1. 针对你所处理的事件，找出合适的 XyzListener 接口
2. 编写一个新的类，这个类实现了 XyzListener 接口
3. 在你所感兴趣的事件中编写处理代码，而对于那些你不感兴趣的方法可以让它们保持空白（就像实例中的 widgetDefaultSelected()方法）一样

让事件处理更加简单：使用适配器（adapter）

有时候我们可能会感觉这样仍然不够简单，比如我只对 SelectionListener 中的 widgetSelected()方法感兴趣，但是为了能够通过编译器的编译，我却不得不写一个空白的 widgetDefaultSelected()方法（因为 SelectionListener 是一个接口，你必须实现它所有的方法）。

幸运的是，swt 帮我们解决了这个问题，途径就是使用 adapter。在 swt 中，对应于一个 XyzListener 都有一个 XyzAdapter，adapter 都是抽象类并且实现了对应的 listener 接口，它为对应 listener 接口中的每个方法都定义了一个默认实现（基本上就是什么都不做），我们在使用时候只需要 override 掉自己感兴趣的方法就可以了。

结合上一小节中的代码，如果使用 SelectionAdapter 代替 SelectionListener 的话，我们的代码就可以这样写：

```
button.addSelectionListener(new SelectionAdapter(){
        public void widgetSelected(SelectionEvent event) {
           handleSelectionEvent();
        }
     });
```

这样是不是很方便呢？

EventObject：事件处理的附加信息

在处理各种事件时，我们需要一些附加信息，而 EventObject 给我们提供了这些信息。

我们先来看下面这个简单的小程序，在这段程序中，我们创建了两个文本框，当在第一个文本框输入时，第二个文本框中显示输入的字符。

```
1 public class EventDemo2 {
2
3   Text logText;
4
5   public EventDemo2() {
6      Display display = new Display();
7      Shell shell = new Shell(display,SWT.SHELL_TRIM);
8
9      GridLayout layout=new GridLayout();
10      layout.numColumns=2;
11      shell.setLayout(layout);
12      shell.setText("Event demo");
13
14      Label label1=new Label(shell,SWT.RIGHT);
15      label1.setText("text1:");
16      Text text1=new Text(shell,SWT.NONE);
17
18      text1.addKeyListener(new KeyAdapter(){
19         public void keyPressed(KeyEvent e) {
20            Text t=getLogText();
21            String s=t.getText();
22            t.setText(String.valueOf(e.character));
23         }
24      }
25      );
26
27      Label label2=new Label(shell,SWT.RIGHT);
28      label2.setText("text2:");
```

```
29      Text text2=new Text(shell,SWT.NONE);
30      text2.setEditable(false);
31      text2.setBackground(new Color(display,255,255,255));
32      setLogText(text2);
33
34      shell.pack();
35      shell.open();
36
37      while (!shell.isDisposed()) {
38          if (!display.readAndDispatch()) {
39              display.sleep();
40          }
41      }
42      display.dispose();
43  }
44  /**
45   * @param args
46   */
47  public static void main(String[] args) {
48      EventDemo2 demo2=new EventDemo2();
49  }
50  /**
51   * @return Returns the logText.
52   */
53  public Text getLogText() {
54      return logText;
55  }
56  /**
57   * @param logText The logText to set.
58   */
59  public void setLogText(Text logText) {
60      this.logText = logText;
61  }
62 }
63
```

代码段 7

你可能没有兴趣仔细研究这么长的代码，那么让我们只关注这一小段代码：

```
text1.addKeyListener(new KeyAdapter(){
        public void keyPressed(KeyEvent e) {
            Text t=getLogText();
            String s=t.getText();
```

```
        t.setText(String.valueOf(e.character));
    }
}
);
```

在这段代码中，我们使用了 KeyAdapter 来处理键盘事件，而 keyPressed 会在有键按下时候被调用，我们在函数中使用了 KeyEvent 类型的参数 e，并且通过 e.character 得到了按下键对应的字符。

各种 EventObject（例如上面示例中的 KeyEvent）在事件处理函数中作为参数出现，它们可能有不同的属性和方法，利用这些特性我们可以做很多有意思的事情。

我们下面只简单介绍几种 EventObject，它们分别是对应于窗口事件（ShellListener，ShellAdapter）的 ShellEvent，对应于键盘事件(KeyListener，KeyAdapter)的 KeyEvent 和对应于鼠标事件（MouseListener，MouseAdapter）的 MouseEvent。希望可以起到窥一斑而见全豹的作用。

几种 EventObject 简介

ShellEvent

如果你打开 ShellEvent 的 API，你会很惊讶的发现它只有一个布尔型的属性，就是 doit。这个莫名其妙的属性是用来做什么的呢？

我们知道，Shell对应的就是程序的窗口，在ShellListener中定义的几种事件包括窗口激活时候的shellActivated，窗口即将被关闭时候的shellClosed等等。ShellEvent中唯一的属性doit，就是用来设定是否这些动作将有效的。

再说得具体一些，比如Windows下通常我们会通过点击窗口右上角的关闭按钮来关闭窗口，这个时候就会对shellClosed进行调用，如果我们在shellClosed(ShellEvent e)方法中把ShellEvent对象e的doit属性置为了false，那么这次动作就无效，窗口不会被关闭。

在有些其他的 EventObject 中也有 doit 属性，它们的作用都是类似的。比如 KeyEvent 就有这样的一个属性。如果你在 keyPressed 方法中把它置为了 false，就等于你按键盘（对于对应的 widget，也就是 receiver 来讲）没有用。

KeyEvent

其实在前面我们或多或少的已经介绍了一些 KeyEvent 的知识。KeyEvent 包含四个属性：character，doit，keyCode 和 stateMask。

其中 character 我们在前面的示例中使用过，它其实就是按键对应字符，而 doit 和 ShellEvent 中的 doit 含义是相同的。

keyCode是我们称为键码的东西，什么是键码呢？如果你打开 org.eclipse.swt.SWT的API文档，你会发现里面有很多都和键盘有关的整型常量，比如SWT.F1，SWT.F4，SWT.ESC，SWT.KEYPAD_3之类，这就是他们的键码。

而 stateMask 则是用来检测 Alt，Shift，Ctrl 这些键有没有同时被按下。

用 stateMask 与这些键的键码进行位与运算，如果得到的结果不是 0 就说明这些键被按下了，比如如果 stateMake & SWT.ALT 不为零，我们就可以认为 Alt 键被按下了。

MouseEvent

MouseEvent 对应于的是鼠标事件。它同样包含四个属性：button，stateMask，x，y

button 就是说明按下的是哪个键，比如对于普通鼠标来说，1 是左键，2 是右键等等

stateMask 却是用来反映键盘的状态的，这和 KeyEvent 中的 stateMask 含义是相同的。

x 和 y 指的是相对于部件的横坐标和纵坐标。

你可能会觉得有点疑问，光是这么一点属性就能处理鼠标事件了么？如果我有一个滚轮鼠标，那应该用什么事件处理滚轮的动作呢？答案是：目前可能还无法利用事件模式处理，关于这一点可以参照一下这个url:
https://bugs.eclipse.org/bugs/show_bug.cgi?id=58656

关于EventObject我就只介绍到这里，这当然很不够，但是我强烈建议大家在实际应用中多查阅eclipse和swt的相关文档。因为毕竟精力有限，我的目的是让大家通过这篇文章能够找到一个正确获取知识的方向，而不是把这些知识很详细的介绍给大家。

Untyped Events

我们在这里提到了 untyped events，那肯定就有 typed event，typed 和 untyped 本身并不是说事件有什么不一样，而是说事件处理是使用了特定的 Listener 还是没有。我们前面提到的所有事件处理都是 typed 类型，因为它们都使用了特定 Listener。

所谓的 untyped events 你可以理解为一个事件的大杂烩。和 untyped event 相联系的两个类是 Listener 和 Event。在这里我想请大家注意一下，这两个类不是在 org.eclipse.swt.events 中，而是在 org.eclipse.swt.widgets 中。

Listener只有一个方法handleEvent，这个方法里你可以处理任何事件。而如果你打开Event看一下，就能看到我们刚刚在前一小节中介绍过的那些XxxEvent中的属性在这里应有尽有。所以它可以起到替代它们的作用，当然如果是一个窗口被关闭的事件，相信你用keyCode属性意义不大。

让我们看一下下面一段代码

```
1 Shell shell = new Shell ();
2   Listener listener = new Listener () {
3     public void handleEvent (Event e) {
4       switch (e.type) {
```

```
5         case SWT.Resize:
6           System.out.println ("Resize received");
7           break;
8         case SWT.Paint:
9           System.out.println ("Paint received");
10           break;
11         default:
12           System.out.println ("Unknown event received");
13       }
14     }
15 };
16 shell.addListener (SWT.Resize, listener);
17 shell.addListener (SWT.Paint, listener);
18
```

代码段 8

由此我们大体上可以体会到 untyped events 的处理方式。

小结

关于事件处理，我就向大家介绍这么多。到现在为止，我们已经基本上可以写一些简单的 swt 用户交互程序了。然而这还远远不够，毕竟人们总是希望有更华丽（或者说：丰富）的界面，让用户能够获得更好的体验。在下一节中，我计划和大家讨论一些这样的部件。

另外可能你觉得有些疑惑，为什么写了这么多内容，都是关于 swt 的呢？Jface 的内容呢？我的计划是在大部分 swt 有关的内容介绍完了以后再向大家介绍 Jface。事实上，即使不用 Jface，你也完全可以用 swt 构筑起一个非常完美的程序来。

最后，给自己做一个小广告，因为CSDN的blog功能实在太令人失望了，所以我希望在近期内可以把blog迁移到blogjava.net，具体地址是：http://www.blogjava.net/jayliu，欢迎大家提出宝贵意见。

SWT/JFace开发入门指南（五）

使用 Visual Editor 加速你的开发

在eclipse中，你可以使用Visual Editor来进行拖放式的图形界面设计。实际上我觉得在这一方面大家应该可以很容易上手。

如果你安装了 VE 的话，在新建一个 swt 工程的时候你就可以省好多力气了：你不需要找那些名字 n 长的 jar，然后一个一个导入工程的 build path 里面，而是仅仅需要添加一个 User Library，就像下面图示一样：

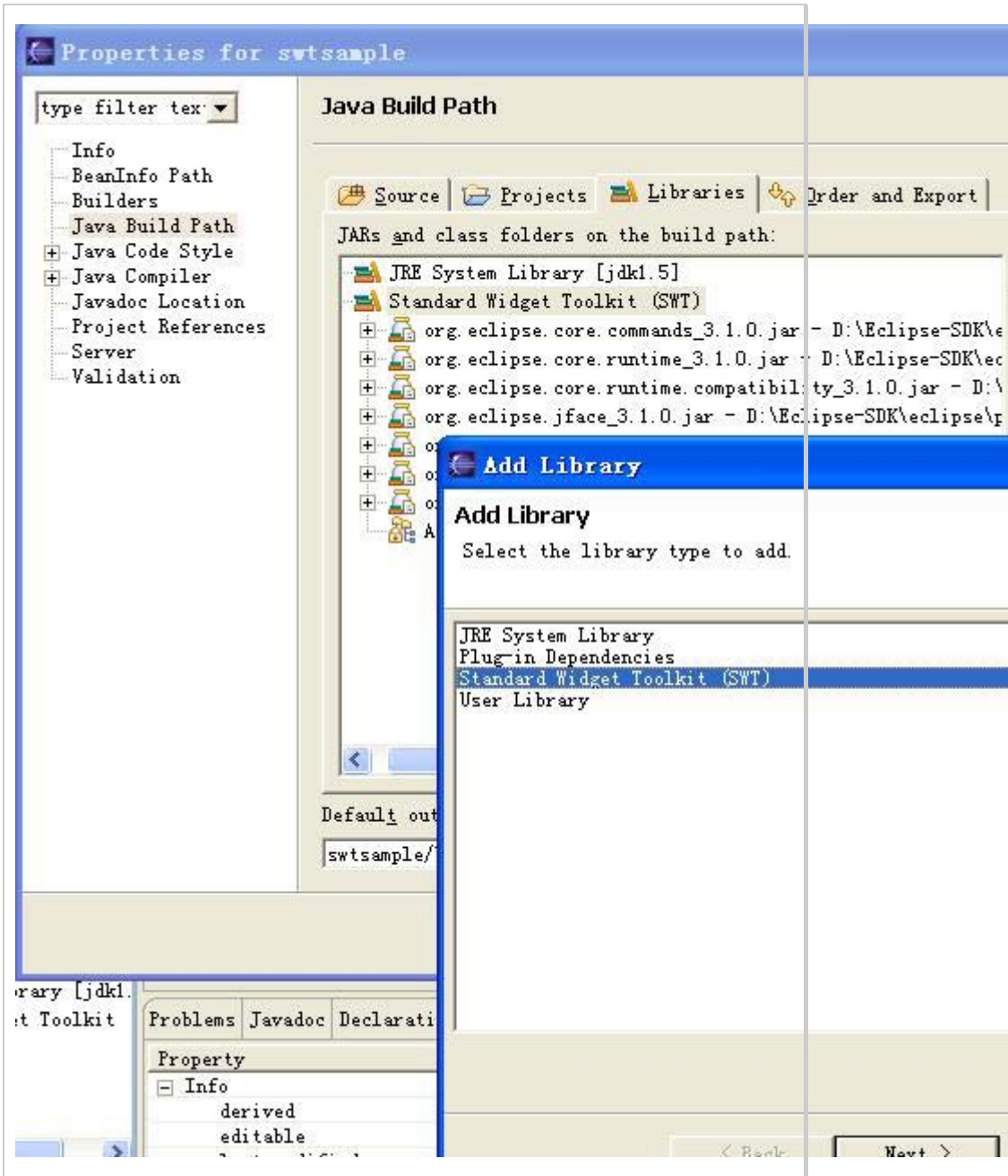

图 7

你可以新建一个 Visual Class。

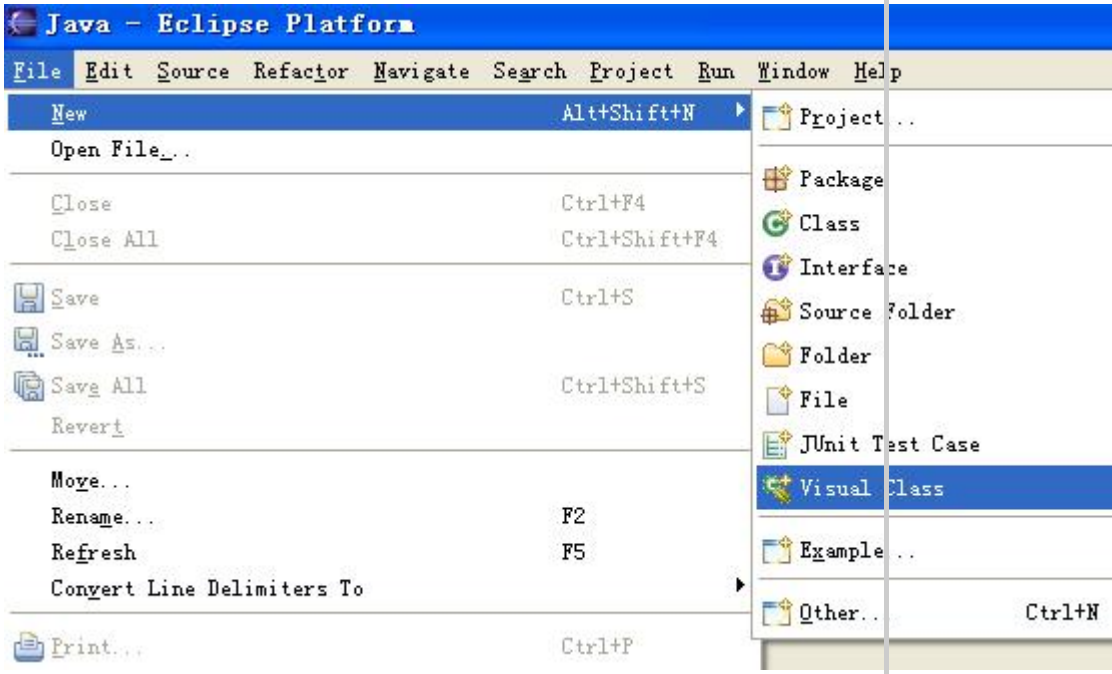

图 8

之后就可以使用 VE 进行可视化的编辑了：

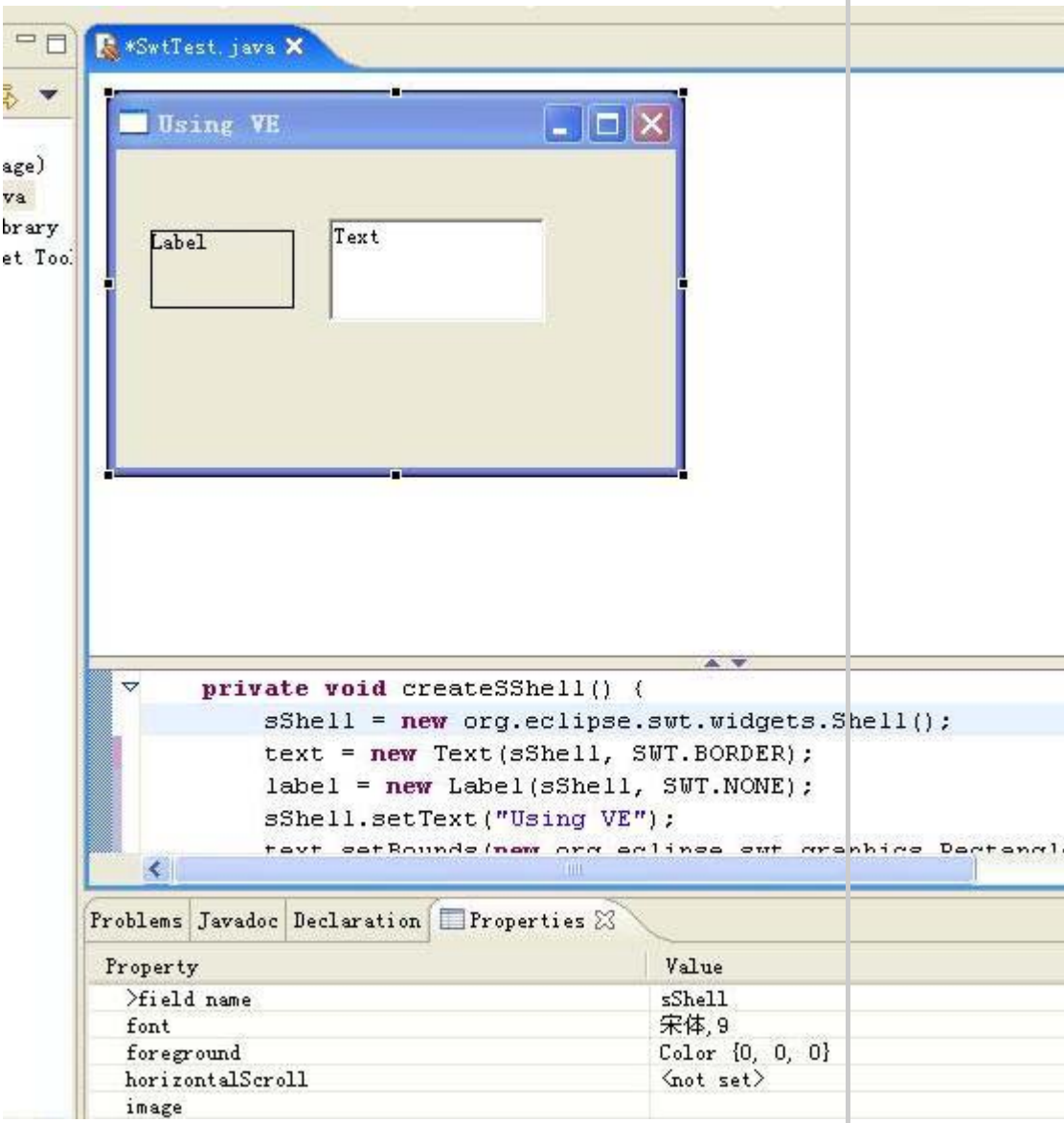

图 9

关于VE的具体应用，我在这里不作具体介绍(说实话，我觉得用起来还是比较简单的)，如果你觉得实在有必要看一篇详细的指南的话，可以参照这篇文章：Extending The Visual Editor
Tutorial: Enabling support for a custom widget

SWT/JFace开发入门指南（六）

使用 Layout 管理 UI 布局

我们在前面在一些例子中已经使用过 Layout 了。那么 Layout 到底是做什么的呢？

我们知道，在设计用户界面时候，我们可以采用的一种办法是手动的为每个部件设置合适当大小和位置。但是这样的话，如果你所要显示的部件比较多，编程量就会非常大，特别是考虑到窗体大小变化时候各种部件的重绘。而实际上，我们可以利用一些通用的规则或者说算法来安排这些部件的排列。比如下图所示的这个窗体：

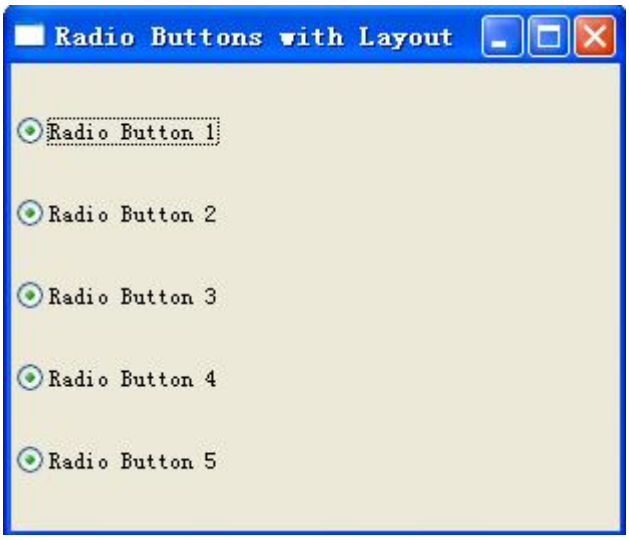

图 10

很显然，对于这些规则化的部件排列，应该有一些更简单的办法，而不是在代码的各个部分写 xxx.setBounds 之类。swt 给我们提供了这样的办法，就是使用 Layout。

你可以把一个 Layout 看成是安排部件位置和大小的一个规则，在应用了 Layout 的 Composite（我们在这里第一次提到了 Composite，Composite 就是一个能够包含其他控件的容器，比如 Shell 就是一个 Composite，我们会在后面的部分详细介绍 Composite）中，所有的子控件都会按照这个规则来进行排列。

在目前（写这篇文章的时候eclipse最新版本是 3.1M7），org.eclipse.swt.layout包中包含四种已经定义好的Layout，它们分别是：FillLayout，FormLayou，GridLayout和RowLayout。我在这里不再一一作介绍，而是推荐大家看一下这篇文章《Understanding Layouts in SWT》。相信如果你有兴趣看完它的话，就会对Layout有比较深的了解。

SWT/JFace开发入门指南（七）

几种特殊的部件

好像因为大家的抱怨比较多，感觉这一段 csdn 的 blog 似乎又有恢复稳定的迹象了，^_^。

前面的两节中，我都没有作一些详细的介绍，而是推荐了两篇文章给大家，可能你会觉得有点不习惯，不过我觉得作为一个软件开发者来说，最重要的一个技能就是你要能够找到自己所需要的资源。而在swt（JFace）开发方面呢，其实如果你能多看一看eclipse的联机帮助和官方网站上的各种文档的话，对你的开发一定会大有帮助。其实我也都是这样一步一步慢慢熟悉了的。

在这一节中，我要向大家介绍几种比较特殊的 widget，包括 Composite，Menu 等。

Composite

其实我们已经接触过 Composite 了，我们前面介绍过 Shell，Shell 就是一种特殊的 Composite。就如它的名字一样，Composite 指的是可以嵌套其他部件的一种部件。

我们先来看一段代码，为了看起来更简明，我把 shell 创建之类的代码都去掉了。

```
Composite composite=new Composite(shell,SWT.V_SCROLL);
composite.setLayout(new RowLayout(SWT.VERTICAL));
Text text=new Text(composite,SWT.NONE);
text.setSize(100,30);
```

代码段 9

这段程序意义不大，不过我们可以看出，Composite 可以作为其他部件的 parent，可以有自己的 Layout。

我们在其中定义 composite 的风格时候用到了 SWT.V_SCROLL，这是用来做什么的呢？它是用来定义这个部件有纵向滚动条的。当然相应当 SWT.H_SCROLL 就定义了部件有横向滚动条。

那么什么样的部件可以有滚动条呢？只要是 org.eclipse.swt.widgets.Scrollable 的子类就可以使用这两个风格定义自己的滚动条。这是我们顺便提到的一点题外话。

如果你把上面的程序填充完整运行一下的话会发现出来的样子很难看。因为 Composite 本身是看不到的。而可以孤零零地看到一个突出的滚动条在那里。所以实际上单独使用 Composite 意义不是很大，我们可以使用它的一些子类。

打开 Composite 的 javadoc，你会发现它的子类实在是太多了。我们只拣两个介绍一下，Group 和 TabFolder。

比起 Composite 来，Group 有一个边框，你可以用 setText 方法设定它的标题。所以这样在视觉上会更好看一些。下面是一个 Group 的图示：

图 11

另外一种特殊 Composite 就是 TabFolder，TabFolder 允许我们定义标签页式的窗体。就像下面图示的那样：

图 12

我们怎么设定各个标签呢？答案是使用 TabItem。可以使用 TabItem 的 setControl 方法设定标签页上的控件。但是这样又出现了一个疑问，setControl 中的参数不是数组，如果我们想设定好几个控件应该怎么办呢？这个时候就用得上 Composite 了。我们可以首先建一个 Composite 实例，然后在里面添加控件，再把这个 Composite 实例本身作为参数传递给 TabItem 的 setControl 方法。

为了更清楚地说明，我们可以看一下下面这段代码：

```
1  public class TabFolderDemo {
2
3    /**
4     * @param args
5     */
6    public static void main(String[] args) {
7        Display display = new Display();
8
9        Shell shell = new Shell(display, SWT.DIALOG_TRIM);
10       shell.setLayout(new RowLayout(SWT.HORIZONTAL));
```

```
11      shell.setText("TabFolder Demo");
12
13      TabFolder tf = new TabFolder(shell, SWT.NONE);
14      tf.setLayout(new FillLayout());
15      TabItem ti = new TabItem(tf, SWT.NONE);
16      ti.setText("A Simple TalItem");
17      Composite composite = new Composite(tf, SWT.NONE);
18      composite.setLayout(new GridLayout(2, true));
19
20      for (int i = 0; i < 3; i++) {
21         Label label = new Label(composite, SWT.RIGHT);
22         Text text = new Text(composite, SWT.NONE);
23         label.setText("Text" + i + ":");
24
25      }
26
27      ti.setControl(composite);
28
29      shell.pack();
30      shell.open();
31
32      while (!shell.isDisposed()) {
33         if (!display.readAndDispatch()) {
34            display.sleep();
35         }
36      }
37      display.dispose();
38   }
39 }
40
```

代码段 10

菜单（Menu）

swt 中和菜单相关的两个类是 Menu 和 MenuItem。顾名思义，Menu 代表的就是整个菜单，而 MenuItem 是菜单中一项一项的实体。打个不恰当的比方来说，Menu 就好比一个书架，而 MenuItem 就是书架上一个一个的格子。

创建菜单

如果我们要创建一个菜单，可以根据下面步骤来进行：

1. 实例化一个 Menu 对象，Menu 有好几种格式的构造函数，你可以选择一个，比如像这样：

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

与其他部件不同的是，Menu 的父控件并不一定需要是 Composite，而是任何一个 Control 都可以。而 Menu 的风格有几种：

- SWT.POP_UP：弹出式菜单
- SWT.BAR：程序中的顶级菜单，比如我们通常看到的包含了“文件，编辑”之类的那个横列的菜单栏
- SWT.DROP_DOWN：下拉式菜单，比如我们在 Word 点击“文件”之后出现的子菜单就是一个典型的下拉式菜单。

注意 Menu 本身是没有 setText 方法的，因为这些具体的显示属性你必须在相应当 MenuItem 中设定。

2. 为 Menu 添加 MenuItem 对象，一个 MenuItem 的构造函数如下所示：

```
MenuItem(Menu parent, int style)
```

或者是：

```
MenuItem(Menu parent, int style, int index)
```

其中 parent 就是所属的 Menu，而 index 可以看成是排列顺序。而 style 可以有以下几种：

- SWT.PUSH：就是一个普通道菜单项，但是这样的 MenuItem 不能有层叠的下一级菜单
- SWT.CASCADE：这样的 MenuItem 可以有层叠的下一级菜单。可以使用 MenuItem 的 setMenu 方法设定它的下一级菜单，作为参数的 Menu 风格必须是 SWT.DROP_DOWN
- SWT. SEPARATOR：分隔符，也就是一道横线，这样的 MenuItem 是没有文字的，所以 setText 不起作用。
- SWT.CHECK：菜单项前面有一个复选标志。
- SWT.RADIO：菜单项前面有单选标志。

下面的图示说明了各种风格不同的 Menu 和 MenuItem

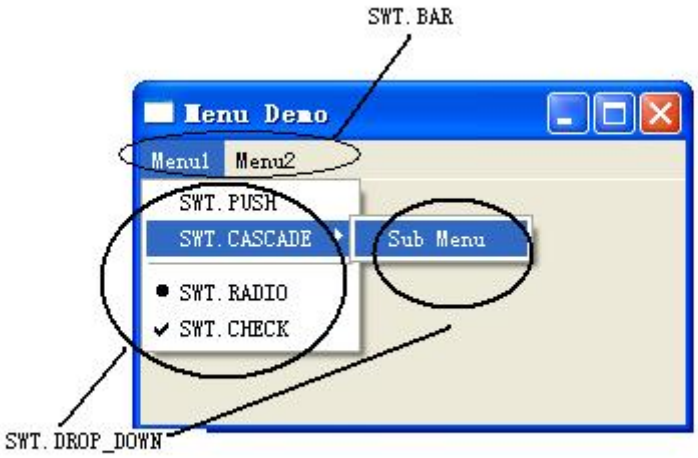

图 13

3. 为有下级菜单的 MenuItem（风格必须为 SWT.CASCADE）设定下级菜单的 Menu 对象（用 setMenu 方法），当然了，你仍然需要为这个下一级的 Menu 设定其中的 MenuItem

4. 如果你的菜单风格式 SWT.BAR，你还需要调用 Shell 的 setMenuBar（参数就是顶级菜单）方法设定窗体菜单。如果是弹出式菜单也就是 SWT.POP_UP，你需要调用所在控件的 setMenu 方法设定弹出菜单为该菜单。

处理菜单事件

菜单的事件处理最主要的还是选择菜单项时候要进行的一些动作，我们可以用 SelectionListener 进行处理。你也可以为菜单展开的事件添加 ArmListener。

下面这段简单的代码展示了一个只有退出程序功能的文件菜单。

```
1 public class MenuDemo {
2
3   private Shell _shell;
4
5   /**
6    * @return Returns the _shell.
7    */
8   public Shell getShell() {
9      return _shell;
10   }
11
12   /**
13    * @param _shell
14    *          The _shell to set.
15    */
16   public void setShell(Shell shell) {
17      this._shell = shell;
18   }
19
20   public MenuDemo() {
21      Display display = new Display();
22      Shell shell = new Shell(display, SWT.SHELL_TRIM);
23      setShell(shell);
24      shell.setLayout(new RowLayout(SWT.HORIZONTAL));
25      shell.setText("Menu Demo");
26
27      Menu menubar = new Menu(shell, SWT.BAR);
28      MenuItem fileitem = new MenuItem(menubar, SWT.CASCADE);
```

```
29      fileitem.setText("&File");
30
31      Menu filemenu = new Menu(shell, SWT.DROP_DOWN);
32      fileitem.setMenu(filemenu);
33      MenuItem exititem = new MenuItem(filemenu, SWT.PUSH);
34      exititem.setText("&Exit");
35      exititem.addSelectionListener(new SelectionAdapter() {
36          @Override
37          public void widgetSelected(SelectionEvent arg0) {
38              MessageBox messagebox = new MessageBox(getShell(), SWT.YES | SWT.NO);
39              messagebox.setText("Exit");
40              messagebox.setMessage("Exit the program?");
41
42              int val=messagebox.open();
43              if(val == SWT.YES)
44              {
45                  getShell().close();
46              }
47          }
48
49      });
50      shell.setMenuBar(menubar);
51      shell.setSize(800, 300);
52      shell.open();
53
54      while (!shell.isDisposed()) {
55          if (!display.readAndDispatch()) {
56              display.sleep();
57          }
58      }
59      display.dispose();
60  }
61
62  public static void main(String[] args) {
63
64      MenuDemo demo=new MenuDemo();
65  }
66
67 }
68
```

代码段 11

加速键

在使用 MenuItem 的 setText 方法时，可以在希望设为加速键的字母前面标&，比如

```
exititem.setText("&Exit");
```

就标明了这个退出菜单项的加速键为 E。

另外也可以使用 setAccelerator 方法设定加速键。

ToolBar 和 CoolBar

在这里，我要向大家说一声 sorry 了，因为我并不打算在这里详细介绍 Toolbar 的各种用法，这主要有两个原因：

1. 工具栏的用法和菜单是类似的，菜单是在实例化一个 Menu 对象以后添加 MenuItem，而工具栏则是在实例化 ToolBar 以后添加 ToolItem（或者实例化 CoolBar 以后添加 CoolItem）。甚至它们的一些风格，事件也很类似。如果你对照着 ToolBar 和 ToolItem（CoolBar,CoolItem）的 API 文档以及前面我对菜单用法的介绍看到话，很快就能比较熟悉工具条的用法。

2. 我们在后面介绍 JFace 的时候，你会发现有更简单的办法创建菜单和工具条。

小结

这一节介绍了几种有限的部件，关于 swt 的介绍到这一节为止就暂时告一段落了。在以后的部分我会开始给大家介绍 JFace。当然了，这并不意味着以后再也没有 swt 的内容介绍给大家了，只是我觉得到现在为止，大家应该已经可以写一些简单的 swt 程序了，我如果再多介绍，就类似于对 API 文档的翻译了。当然如果以后有充足的时间的话，我也希望能够向大家多介绍一些相关的内容。

SWT/JFace开发入门指南（八）

JFace 以及其他

关于 JFace：一个简单的介绍

我们已经有了 swt，我们用 swt 可以写出一个完整的程序来，那么我们为什么需要 Jface 呢？

对于这一点，本文作者（就是我了，嘿嘿）的理解是：使用 JFace 比只是单纯地使用 swt 编程更加简单，或者说：代码量更少。毕竟，你完全可以用汇编写一个用户界面，但是付出的代价似乎大了一点:P。

如果你在使用 swt 编程，那么 JFace 的知识并不是必需的：你完全可以不用 JFace 就可以写出任何你需要的功能。但是如果你使用 JFace，你必需对 swt 有一些了解，因为 JFace 需要 swt 的各种部件构建用户界面。

我觉得我们可以在某种程度上这样看 JFace：它封装了一部分 swt 的功能，所谓“封装”可以从几个方面来看：

首先，你可以使用 JFace 的某些机制来代替 swt 中的一些机制

其次，JFace 中各种功能的实现都是依赖于底层的 swt 的。

最后，你可以在使用 JFace 时候同时使用 swt。

这篇文章的组织结构

在这篇文章以后的部分，我将会向大家介绍以下的内容：

首先，我会从一个简单的示例程序开始展示如何开始写一个 JFace 程序

之后我会向大家介绍 JFace 的事件模型（与 swt 的事件模型不同）

然后我会向大家介绍与构建 JFace 用户界面相关的一些知识。

目前来讲，因为我刚刚写到这里，这是我所能想到的一些部分，当然，可能在以后的文章中略有不同。

另外的参考资料

在这一系列文章的第一节（http://blog.csdn.net/jayliu/archive/2005/04/29/367757.aspx）中，我向大家介绍了一些参考资料。现在向大家再介绍一篇在IBM developerworks 上发现的一篇很好的文章：

在 eclipse Workbench 之外使用 eclipse GUI，这篇文章共有三部分，地址列在下面：

http://www-128.ibm.com/developerworks/cn/linux/opensource/os-ecgui1/index.html

http://www-128.ibm.com/developerworks/cn/linux/opensource/os-ecgui2/index.html

http://www-128.ibm.com/developerworks/cn/linux/opensource/os-ecgui3/index.html

环境的配置

关于如何配置编程环境，可以参照这一系列文章的第一篇（http://blog.csdn.net/jayliu/archive/2005/04/29/367757.aspx），在这里我不再赘述。

SWT/JFace开发入门指南（九）

JFace 的 Hello,world!

我们仍然是从一个最简单的 Hello,world!开始介绍 JFace。为了更形象一些，首先把程序列出来：

```
1
2 public class HelloJface extends Window {
3
4   public HelloJface(Shell arg0) {
5      super(arg0);
6   }
7    @Override
8   protected Control createContents(Composite parent) {
9      Text text = new Text(parent, SWT.NONE);
10      text.setText("Hello,world!");
11      return parent;
12
13   }
14   /**
15    * @param args
16    */
17   public static void main(String[] args) {
18
19      HelloJface demo = new HelloJface(null);
20      demo.setBlockOnOpen(true);
21      demo.open();
22      Display.getCurrent().dispose();
23
24   }
25 }
26
```

代码段 12

首先我们从这段代码来看一下使用 JFace 和单纯地使用 swt 写程序有什么不一样：

在 swt 程序中，我们需要自己创建 Display，自己创建 Shell，但是在这里，我们只需要：

创建一个继承自 Window（org.eclipse.jface.window.Window）的类

在这个类的 createContents 方法中为窗口添加部件

将这个对象的 blockOnOpen 属性设定为 true，这个属性的含义就和它的名字一样，窗口会一直保持打开的状态（接收各种事件）直到被关闭。

调用这个对象的 open 方法即打开了窗口

由于设定了 blockOnOpen，窗口会保持接受各种事件，知道用户（或者程序）关闭了它。

在关闭以后，程序继续向下运行，我们需要将资源释放掉，所以有了这样一句话：

```
Display.getCurrent().dispose();
```

其中 Display.getCurrent()得到了程序的 display 对象，并进而调用 dispost()方法释放了各种资源。

其实这也是我们写一个 JFace 程序一般的步骤，当然我们可能还会添加事件处理之类，但是大体上都是这样的。

因为这个程序运行的结果其实和我们以前 swt 的 Hello,world!是一样的，所以在这里我也不再贴图了。大家可以自己运行看一下。

Window，ApplicationWindow，Dialog

关于 Window 的 Q&A

JFace 中的 Window 就是一个窗口。我们知道在 swt 中窗口是用 Shell 表示的，而 JFace 中的 Window 其实也可以看作是对于 Shell 的一种封装。

在 org.eclipse.jface.window.Window 中，有几个方法需要我们注意，为了更加突出各自的功能，我把这些介绍写成 Q&A 的形式：

Q：如何为窗体添加各种功能部件？

A：方法就是重载 createContents 方法，这个方法中我们可以给窗体中创建一些子部件，比如 Text,Lable,Composite 之类，在前面的 demo 中我们是只创建了一个文本框。

Q：如何定义窗体的风格？

A：可以通过调用 setShellStyle 方法来设定窗体的风格，比如如果我们在前面程序的 main 函数中添加这样一句话（在 demo.open();之前）：

```
demo.setShellStyle(SWT.DIALOG_TRIM);
```

那么出现的就是一个对话框风格的窗体。

Q:如何定义窗体的 Layout，标题等属性

A：可以通过重载 configureShell 方法实现，比如下面这样一段程序：

```
@Override
  protected void configureShell(Shell shell) {

    shell.setText("JFace Window");
    shell.setLayout(new RowLayout(SWT.VERTICAL));
  }
```

就设定了窗口标题为“JFace Window”，Layout 也设定为 RowLayout。

默认窗体的 Layout 为 GridLayout。

这个简短的 Q&A 就到这里为止。对于 Window 的介绍也到这里告一段落，下面我们简单地看一下 Window 的两个子类：ApplicationWindow（org.eclipse.jface.window.ApplicationWindow）和 Dialog（org.eclipse.jface.dialogs.Dialog）

ApplicationWindow

ApplicationWindow 除了具有 Window 的特性以外，还允许我们方面地添加菜单，工具条（Toolbar 或者 Coolbar），状态条之类的。关于这方面的特性我会在后面的文章中做单独介绍，这里就不再赘述了。

对话框

JFace 中的 Dialog 是一个抽象类，所以你必须使用它的子类或者自己写一个它的子类。当然了，实际上你根本没有必要去自己继承。

离题万里：swt 中的 MessageBox

关于对话框，首先我要给大家介绍的却是一个 swt 中的 MessageBox（org.eclipse.swt.widgets.MessageBox）。实际上，如果你看过这个系列文章中介绍 swt 事件模式的内容的话，可能早就注意到我在里面使用过 MessageBox 了。

Swt 中的 MessageBox 允许我们通过指定风格来改变对话框的外观，比如如果对话框的风格中包含 SWT.OK，它就会有一个确定按钮；如果包含 SWT.CANCEL，就会有一个取消按钮，如果包含 ICON_QUESTION 那么弹出的对话框就有一个问号图标，等等吧。

比如我们看下面这段程序：

```
MessageBox dialog=new MessageBox(shell,SWT.OK|SWT.ICON_INFORMATION);
dialog.setText("Hello");
dialog.setMessage("Hello,world!");
dialog.open();
```

代码段 13

这里我们从构造函数的 style 参数可以看出，这是一个带有信息图标（一个感叹号）和一个确定按钮的对话框。具体图示见下面：

图14

对比：JFace 中的 MessageDialog

前面一小节中，我简单介绍了一下 swt 中的 MessageBox。在 JFace 中，我们使用 MessageDialog（org.eclipse.jface.dialogs.MessageDialog）来实现类似的功能。

如果要实现我们在上面所演示的那样一个带信息提示和确定按钮的对话框，我们只需要这样写：

```
MessageDialog.openInformation(shell,"Hello","Hello,world");
```

运行出来以后，对话框的大小可能和 swt 中的有一些区别。不过这样是不是很方便呢？其实 JFace 能做的事情通过 swt 编程也都可以做到，不过一般说来，JFace 都简化了编程。

在 MessageDialog 中，类似于 openXxxx 的静态方法还有好几个，比如：

openWarning：打开一个带警告图标的对话框

openConfirm：打开一个只有确定和取消按钮的对话框

…

不过你也许注意到了，这些静态方法返回类型都不一样，有的是 void,有的是 boolean，这些返回值（如果有的话）就反应了用户选择了哪个按钮。比如 openConfirm，如果按了确定，那么返回的就是 true。

输入框：InputDialog

相对来说，输入对话框的用法可能会比较负责一些，你必须 InputDialog 对象，然后调用它的 open 方法打开对话框，获得这个方法的返回值（以确定用户是点击了确定还是取消）。然后再通过 getValue()方法获得用户的输入。

为了更加形象，我们举一个例子。我们还是用前面的 Hello,world!程序，不过把它的 createContents 方法改成下面一段代码：

```
1 @Override
2    protected Control createContents(final Composite parent) {
3        Button button = new Button(parent, SWT.NONE);
```

```
4       button.setText("Click me!");
5       button.addSelectionListener(new SelectionAdapter() {
6           @Override
7           public void widgetSelected(SelectionEvent arg0) {
8
9               InputDialog input = new InputDialog(parent.getShell(),
10                      "Input Dialog Title", "Please input some string here:",
11                      "initial value", null);
12              if(input.open()== Window.OK)
13              {
14                  System.out.println(input.getValue());
15              }
16          }
17
18      });
19      return parent;
20  }
21
```

代码段 14

运行之后，界面如下面图示：

图 15

如果按了确定的话，控制台会显示你输入的信息

小结

因为前一段有些事情，所以都一直没有来得及写这个系列的文章。今天贴出的这篇文章中，通过一个简单程序演示了一个 JFace 程序的基本写法，同时也顺便提了一

下 Window 和它的两个子类。关于对话框我只向大家介绍了两种最常用的，其实还是有很多的，这需要大家在实践中不断探索。

SWT/JFace开发入门指南（十）

JFace 中的事件模式

大家好，因为工作的事情搞了一个多月，现在终于暂时安定下来了。这一系列的文章我也会继续往下写。

在这一节中，我会向大家介绍JFace中的事件模式。其实我相信这篇文章的读者应该大部分都会接触eclipse，这样可能也会接触过eclipse的插件开发。就是没有接触过，大家也可能会有在eclipse里面新建工程的时候出于各种原因（比如好奇心）点了plug-in project的时候吧。其实作为一个程序员来讲，保持好奇是很重要的。如果你大概看过一个plug-in project的结构，虽然可能不能全部理解，但是我相信也应该对Action之类有一些了解。我们这一节主要就是围绕Action来写的。为了增加可读性，我们首先介绍几个名词，这些名词都可以从eclipse的文档中找到。

什么是 Action

JFace 中的一个 Action 可以简单地理解成一个命令。那么它和事件有什么关系呢？比如说我点了一个菜单，那么点击本身就是一个事件，但是这个事件的影响就是相应的命令被执行了。大家日常使用的一些软件比如 Office 都是有菜单和工具栏的，而一个菜单项和一个工具栏可能执行的是同一个命令。比如 Word 里面要新建一个文档的话可以通过“文件”菜单下的“新建”实现，也可以通过点击工具栏上相应的图标实现。这个新建地功能本身在 JFace 里面是可以使用 Action 来实现的。

在 JFace 里面，Action 可以关联到菜单，工具条，以及按钮（也就是 Button）。当然关于如何关联，我们会在后面向大家详细介绍。

Action 在 JFace 里面的定义是一个接口 org.eclipse.jface.action.IAction。当然实际上你写程序的时候必须自己来实现这个接口，写出自己的 Action 类来。

IAction 里面最重要的方法是 run()，它也是事件触发以后执行的代码。其他的方法都是一些辅助性的方法，不是我们要关注的重点。为了能够将精力集中在我们所关注的事情上，通常我们不是实现 IAction 接口，而是通过继承 org.eclipse.jface.action.Action 这个抽象类来实现 Action。下面我们通过一个例子来说明 Action 的用法。

Hello,Action!

首先我们先不管用户界面，先定义一个最简单的 Action 类。

```
1 public class HelloAction extends Action{
2     private Shell shell;
3 
4     public HelloAction(Shell shell) {
5         super("&Hello",Action.AS_PUSH_BUTTON);
6         this.shell=shell;
```

```
7    }
8
9
10    @Override
11    public void run() {
12        MessageDialog.openInformation(shell, "Hello", "Hello,Action!");
13    }
14
15 }
16
```

代码段 15

这段代码其实应该还是很好读懂的。带参的构造函数带进来一个 Shell 实例，而 run()方法说明了这个 Action 的功能就是显示一个对话框。第 5 行中的代码调用了父类的构造函数，其中第一个参数是 Action 对应的文本，前面的&符号表明了 H 是热键，而第二个参数则是一个风格参数。如果大家继续向后看的话，就会发现这个 Action 被附加在了一个按钮上面，而按钮上显示的文本就是 Hello，如果你定义的风格不是 AS_PUSH_BUTTON 而是 AS_RADIO_BUTTON 的话就会发现按钮已经不是一个纯粹的按钮了，而是一个单选钮。相应的其他风格可以参照 Javadoc。

```
1
2 public class HelloJface extends ApplicationWindow {
3    public HelloJface(Shell shell) {
4        super(shell);
5    }
6      @Override
7    protected Control createContents(Composite parent) {
8         HelloAction action=new HelloAction(parent.getShell());
9         ActionContributionItem aci=new ActionContributionItem(action);
10         aci.fill(parent);
11         return parent;
12   }
13   /**
14    * @param args
15    */
16   public static void main(String[] args) {
17
18       HelloJface demo = new HelloJface(null);
19       demo.setBlockOnOpen(true);
20       demo.open();
21       Display.getCurrent().dispose();
22
23   }
24 }
```

代码段 16

和前面一节的代码相比，我们只是修改了 createContents 方法。首先创建了一个 HelloAction 的实例，然后又创建了一个 ActionContributionItem 的实例，最后调用了这个实例的 fill 方法将按钮添加到窗口中，这就是全部了。是不是很简单呢？程序运行出来的效果如下图：

图 16

可能看了这个例子，你会认为 ActionContributionItem 这个类表示的就是一个按钮了。但是实际上并不是的，它在图形界面上表示成什么样子，随着不同的 fill 调用又有不同。在下一节中，我会向大家深入介绍 Contribution Item 以及 JFace 中的菜单，工具条等的应用。这一节就到这里结束了，因为刚刚换了工作环境，有很多事情需要去做，所以写得比较短，请大家见谅：）。

SWT/Jface开发入门指南(十一)

JFace 中的工具条和菜单

前一节中我们简单介绍了一下 Action。其实所谓的 Action 就是一个最常用的事件，举个例子来说，对于一个按钮来说它可以有多个事件，比如按键，焦点，鼠标，等等等等吧，但是实际上在使用程序的时候，我们最关心的，就是按下去这个按钮会发生什么，这个其实就是所谓的 Action。如果大家以前做过 swing/awt 变成的话，应该对 Action 不会陌生。

在 JFace 里面，一个 Action 可以对应多个 GUI 对象，这些对象就是所谓的 Contribution Item。比如我们在一般程序里面很常见的“文件”菜单，下面都会有“新建”，“保存”等等。同时我们可以在工具条上放置相应的按钮，那么这些都是有相同的功能，在 JFace 里面我们可以只写一个 Action，然后把它映射到不同的 ContributionItem 去，而不必为每个部件都写一串处理事件。

我们下面还是通过一个简单的例子来说明，在 JFace 中怎么使用菜单和工具条这两种最基本也是最有用的 Contribution Item。

我们这个程序写得很傻，就是一个光秃秃的窗口上做了一个菜单和工具条按钮，功能也只有一个，就是每次点一下，就弹出一个输入框来问你名字是什么，然后显示一个 Hello, xxx 之类的。

首先我们还是来写一个 Action 类：

```
1
2 public class SayHiAction extends Action {
3   private Shell shell;
4
5   public SayHiAction(Shell shell) {
6     super();
7     this.shell = shell;
8     this.setText("Say&Hi@Ctrl+H");
9   }
10
11   @Override
12   public void run() {
13     InputDialog input = new InputDialog(shell, "Input your name",
14         "Please input your name here:", null, null);
15     if (input.open() == Window.OK) {
16       MessageDialog.openInformation(shell, "Hello", "Hello, "
17           + input.getValue() + "!");
18     }
19
20   }
21
```

```
22 }
23
```

代码段 17

这只是一个很简单的 Action 类，没有太多可说的。

然后我们创建一个 ApplicationWindow 类：

```
1
2 public class Hiyou extends ApplicationWindow {
3
4   private SayHiAction hiaction;
5   public Hiyou(Shell parentShell) {
6     super(parentShell);
7     hiaction=new SayHiAction(getShell());
8     addMenuBar();
9     addToolBar(SWT.FLAT | SWT.WRAP);
10   }
11
12   @Override
13   protected ToolBarManager createToolBarManager(int style) {
14     ToolBarManager toolbar=new ToolBarManager();
15     toolbar.add(hiaction);
16     return toolbar;
17   }
18
19   @Override
20   protected MenuManager createMenuManager() {
21     MenuManager menubar=new MenuManager();
22     MenuManager fileMenu=new MenuManager("&File");
23     fileMenu.add(hiaction);
24     menubar.add(fileMenu);
25     return menubar;
26   }
27
28   /**
29    * @param args
30    */
31   public static void main(String[] args) {
32     Hiyou window=new Hiyou(null);
33     window.setBlockOnOpen(true);
34     window.open();
35     Display.getCurrent().dispose();
```

```
36    }
37
38 }
39
```

代码段 18

大家可能已经注意到了，在这里面我们重载了 createMenuManager 和 createToolBarManager 两个方法，它们的用途就和名字一样，一个是用来创建菜单的，一个是用来创建工具条的。重载了这两个方法以后，通过在构造函数中调用 addMenuBar 和 addToolBar 让工具条和菜单显示出来。

这里值得一提的是 MenuManager 和 ToolBarManager 类，如果大家翻一下 API 文档的话会发现它们都是所谓的 contribution manager（实现了 IConntributionManager 接口），你可以通过这些 contribution manager 来实现对特定组件的管理（添加删除等等）。

具体到菜单的创建，看了我们上面的代码就很明白了，就直接调用相应 MenuManager 的 add 方法把 action 添加上就可以了。JFace 会自动找到这个 Action 的 getText 方法设置菜单的文字。如果是有好几层菜单，那么只要在重新 new 一个 MenuManager 添加到已有的 MenuManager 里面就可以了。就象前面代码中的：

```
menubar.add(fileMenu);
```

至于工具条就更简单了，创建一个 ToolBarManager 然后直接 add 对应的 Action 就可以了。

添加图标

如果菜单只是文字还没有什么，如果你的工具条都是文字是不是会显得干巴巴的？其实只要我们为 Action 设置 ImageDescriptor 就可以了，比如你可以自己画一个图标保存到 Action 的包下面（我画了一个 hi.gif），然后把 Action 的构造函数改写成这样：

```
public SayHiAction(Shell shell) {
      super();
      this.shell = shell;
      this.setText("Say&Hi@Ctrl+H");
      this.setImageDescriptor(ImageDescriptor.createFromFile(this.getClass(),
            "hi.gif"));
   }
```

大家注意最后一句话，就是为 action 设置图标的。然后再运行一下就会发现菜单和工具栏都有图标了。

在这里给一个社区做个广告，大家可以登陆到在中国eclipse社区（http://www.eclipseworld.org），在那里你也可以找到很多的帮助和支持，当然，你也可以在那里找到我。