5.1.1 Eclipse 插件开发简介

插件的概念读者应该很熟悉，象 MP3 播放软件 WINAMP 的皮肤插件、Windows Media Player 的众多的外观插件、音效插件等等。但如果你以为插件只能做成为原软件的边角料，那是可以理解的，因为你还没有看到过 Eclipse 的插件是什么样的。Eclipse 可以全面更新你对插件的概念，它也是对插件概念运用得最彻底最炉火纯青的一个软件。

在第一章我们就介绍了 Eclipse 的技术特点，Eclipse 的内核很小，其他功能都是基于这个内核上的插件，如 Eclipse 自带的 UNIT、ANT 等。而且 Eclipse 还开放了自己的插件机制，并提供了很好的插件开发环境，让用户可以自己来开发 Eclipse 的插件。想知道开发 Eclipse 的插件能到什么程度吗？看看这些 Eclipse 上的插件吧：用于 UML 建模的 Together for Eclipse、用于 JSP 的 MyEclipse 和 Lomboz、IBM 的全能开发工具 WSAD 等等，它们全是 Eclipse 的插件。如果微软愿意，也可以把 Office 软件做成 Eclipse 的插件。如果 Adobe 有兴趣，Photoshop 也可以有 for Eclipse 的插件版，Eclipse 中的 API Draw2D 的绘图功能也是很功的。

Eclipse 的各式插件正如雨后春笋般不断冒出，Eclipse 已经超越了开发环境的概念，它的目标是做成一个通用的平台，让尽量多的软件做为插件集成在上面，成为未来的集成的桌面环境。同样我们可以将我们的应用系统写成 Eclipse 插件，笔者就在 2004 年参与开发了一个项目管理软件，该软件就是以 Eclipse 的插件形式开发的。

5.1.2 Eclipse 插件开发的优势和不足

那么将软件写成插件有什么好处呢？对于用户来说 Eclipse 的使用环境比较友好，前面介绍的 SWT/JFace 中还是比较基本的界面元素，象 Eclipse 中的视图、编辑窗、停泊窗这些界面如果实现呢？如果用 Appliction 的方式会很麻烦，如果写成 Eclipse 插件则实现这些界面风格不会吹灰之力。可以说把软件开发成 Eclipse 插件的最大好处就是界面风格友好统一，如果用户较熟悉 Eclipse 操作的话这种优势就更明显。

当然将软件写成插件形式也有一定的缺陷。首先插件必须依附 Eclipse，如果要安装插件就得先安装 Eclipse。其次，插件和 Eclipse 融合在一起，原 Eclipse 的一些菜单和工具栏是无法完全屏蔽的。

5.2 插件的 Hello World

5.2.1 使用向导一步步创建 HelloWorld

我们利用 Eclipse 的“新建”向导来创建一个简单的插件。

1、新建一个插件项目

（1）选择主菜单“文件→新建→项目”，在弹出的窗口中（如图 5.1 所示）选择“插件开发”下的“插件项目”，然后单击“下一步”。

图 5.1 项目类型选择

（2）如图 5.2 所示，输入项目名“myplugin”，其他设置不变，然后单击“下一步”。

图 5.2　项目名称

（3）在新显示的窗口中接受所有缺省值不变，直接单击“下一步”，这时将显示模板选择窗口（如图 5.3 所示）。勾选“使用其中一个模板来创建插件”项，然后选择模板“Hello,World”项。最后单击“完成”结束向导对话框。

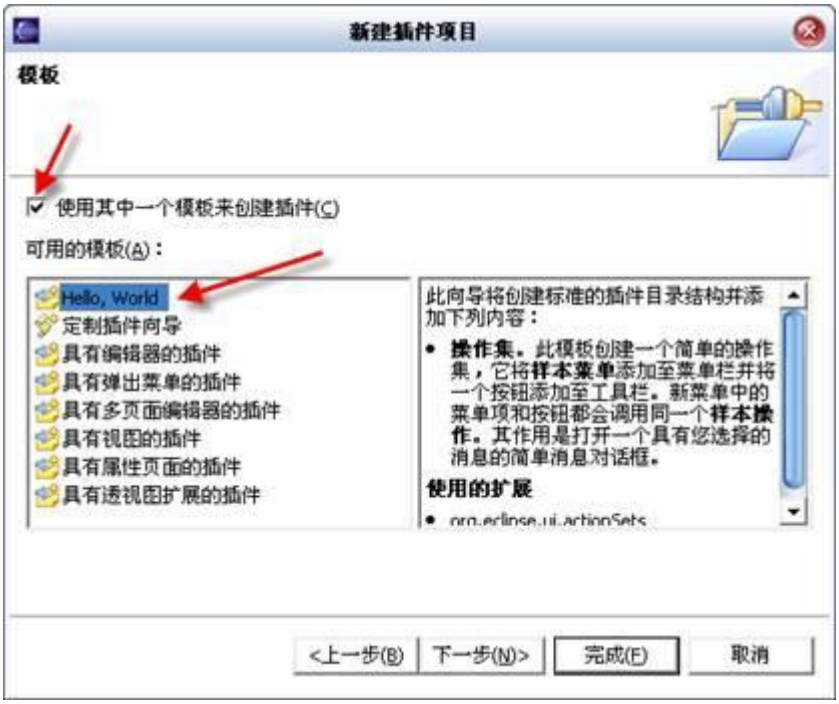


图 5.3　模板选择窗口

2、插件项目 myplugin 简介

如果在新建项目中操作正确，Eclipse 将显示如图 5.4 所示界面。

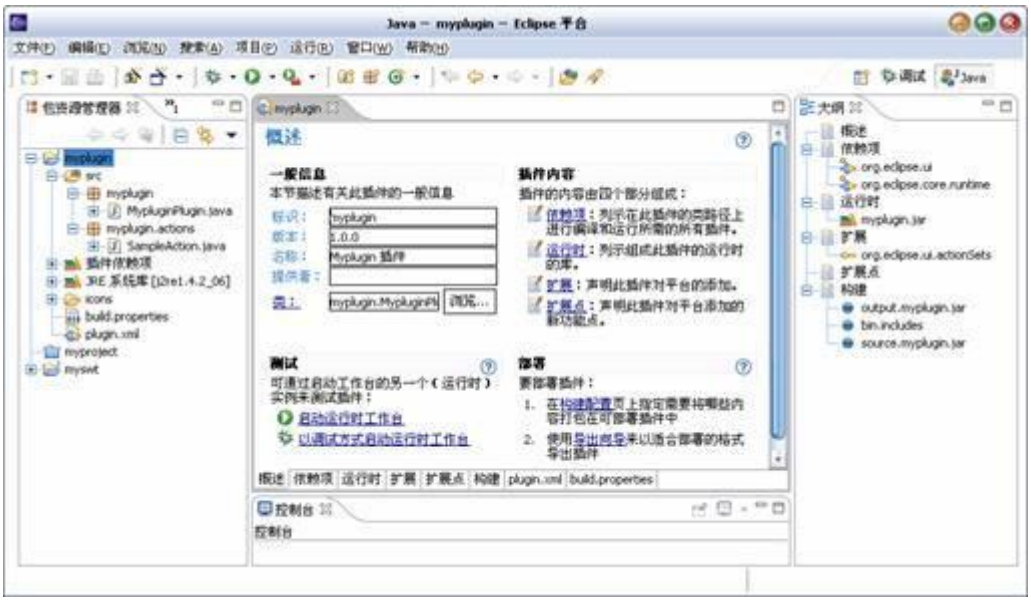


图 5.4　建立一个插件项目后的 Eclipse 界面

界面的左边视图中多了一个名为“myplugin”的项目。项目中有两个文件：MypluginPlugin.java、SampleAction.java。MypluginPlugin.java 较重要，今后将会使用到它，而 SampleAction.java 则是一个类似 JFace 中的 Action，可以把它看做是插件中的 Action，等会运行时我们将看到 SampleAction.java 的效果。

项目根目录下还有一个非常重要文件的 plugin.xml，这个文件是插件的入口文件，Eclipse 是根据这个文件里的设置信息来加载插件的。在插件开发初期会频繁在这个文件中做编辑，术语叫“设置扩展点”。象在 Eclipse 的增加主菜单、视图、按钮等，都是在这个文件里面设置不同的扩展点，后面的将详细讲到如何编辑此文件。有人会问：开发一个系统会有很多的菜单和按钮，是不是都要在这个文件里设置呢？回答：不必。在 plugin.xml 里只设置和 Eclipse 接壤的主要扩展点，其他软件自有的菜单和按钮不用在 plugin.xml 设置了。图 5.4 的 Eclipse 界面中部显示的就是 plugin.xml 的设置窗口，单击该窗口下部的 plugin.xml 项后（如图 5.5 所示），就可以直接编辑此文件。

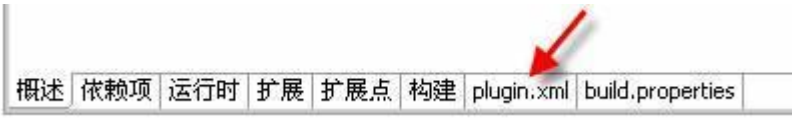


图 5.5　plugin.xml 编辑窗下部的选项条

3、运行插件

如图 5.6 所示，选择主菜单“运行→运行方式→运行工作平台”，这种是专用是插件的运行方式，它将打开一个新的 Eclipse 环境，并同时将插件项目编译加载到新的 Eclipse 环境

中。今后开发经常要通过这个方法来试运行所开发的插件项目，不过那时候选择"运行→调试方式→运行工作平台"以调试方式来运行插件会比较多，Eclipse 支持调试期间的热修改，不用每次修改都新启一个 Eclipse，这样能节省很多调试开发时间。

新开的 Eclipse 界面如图 5.6 所示，在新的 Eclipse 环境中新增加了一个工具栏按钮和一个主菜单项。单击此按钮或菜单项，将弹出一个"Hello，Eclipse world"信息提示框。

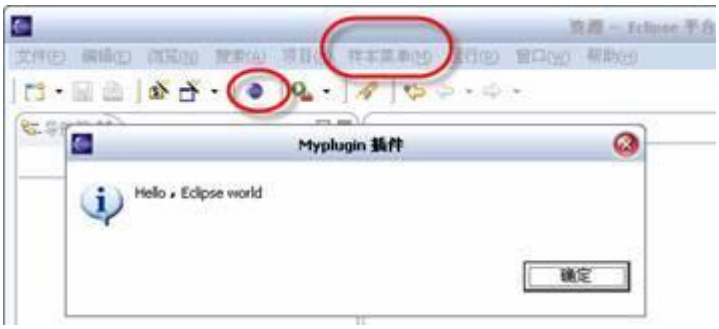


图 5.6　myplugin 插件运行效果图

4、总结

本节里我们还只是依样画葫芦，感觉有点云里雾里的吧。但不管怎么样，第一个 Eclipse 插件已经在我们手里诞生了，下一节我们将不用 HelloWorld 模板来新建一个空白的插件项目，然后一步步的经过手工实现这个 Hello World 插件项目所拥有的功能。

5.2.2　以空白项目为基础手工创建 HelloWorld

1、新建项目

按照上一节所讲新建插件项目的方法，新建一个名为 myplugin2 的插件项目。注意在最后一步不要选择任何模板，直接单击"完成"结束向导对话框，除此之外的其他步骤都一样。很幸运，Eclipse3.0 修正了很多 BUG，象以前用 Eclipse2.X 中文版时，在这一步还会出很多库引用的错误，要很麻烦的一个个去修正。

2、创建 IWorkbenchWindowActionDelegate 接口的实现类

新建一个包 book.chapter_5，并将上一节中由 HelloWorld 模板生成的 myplugin 项目中的 SampleAction.java 文件复制到本项目中（Eclipse 支持鼠标拖拉操做）。然后对 SampleAction 做了一些小修改：删除了无用的注释和构造函数，修改了一下弹出框的提示文字，修改后的代码如下：

```
/**
 * 本类相当于插件的 Action，要在 Eclipse 中增加主菜单或工具栏按钮，
 * 就需要写一个实现 IWorkbenchWindowActionDelegate 接口的类
```

```
 */
public class SampleAction implements IWorkbenchWindowActionDelegate
{
    private IWorkbenchWindow window;
    public void run(IAction action) {
        //打开一个信息提示框
        MessageDialog.openInformation(window.getShell(),
                                    "Myplugin2 插件", "Hello，这是手
工做的插件");
    }
    public void selectionChanged(IAction action, ISelection
selection) {}
    public void dispose() {}
    public void init(IWorkbenchWindow window) {this.window = window;}
}
```

3、原 plugin.xml 文件各设置项说明

如图 5.7 所示，将 plugin.xml 文件打开，并单击窗口下的“plugin.xml”项转到其代码编辑窗。

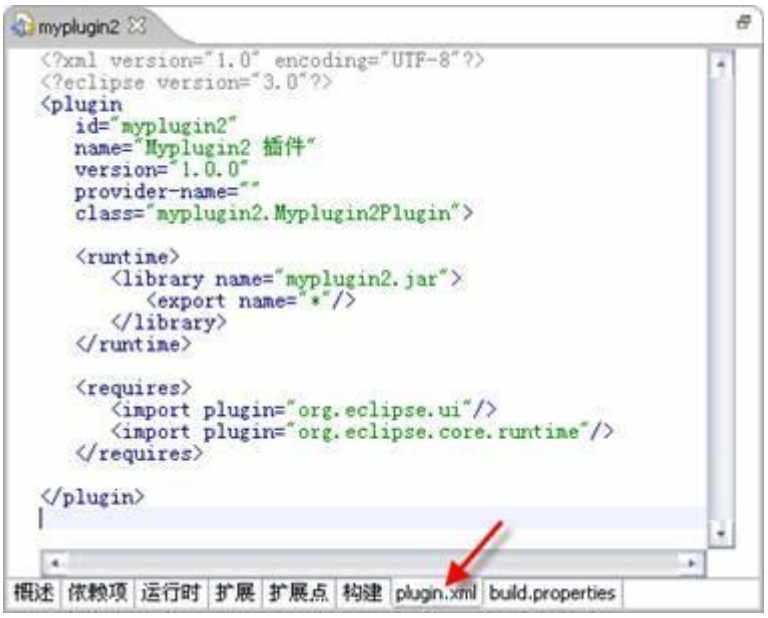


图 5.7　plugin.xml 的代码编辑窗

项详细介绍其中的各项设置如下:

（1）<plugin>项

```
<plugin
   id="myplugin2"
   name="Myplugin2 插件"
   version="1.0.0"
   provider-name=""
   class="myplugin2.Myplugin2Plugin">
```

说明: <plugin>是 plugin.xml 的主体。

id - 插件的唯一标识。实际项目中一般加上包名或网址名来命名 id，比如 eclipse 的 tomcat 插件是这样命名的: org.eclipse.tomcat，这样在世界上就不会有插件的标识名和你重名了。以后在某些扩展点中的属性也会用到标识符作为名称的前缀。

name - 插件的名称，可以不唯一。

version - 插件版本号。

provider-name - 插件开发商的名称，可以写上作者或公司的名称。

class - 插件类的名称，即插件项目自动生成的 MypluginPlugin2.java 文件的类，前面加上包名。

（2）< runtime>项

```
<runtime>
  <library name="myplugin2.jar">
     <export name="*"/>
  </library>
</runtime>
```

说明: 这里是声明插件运行时需要的 jar 包，比如插件要连接 MySQL 数据库需要它的一个包，如下定义，其中"lib\"是该包所在路径。其中本插件自身的 jar 包也要声明，而且本插件在打包时将以 myplugin2.jar 为名打包。

```
<runtime>
```

```
  <library name="myplugin2.jar">
     <export name="*"/>
  </library>
  <library name="lib\mysql-connector-java-3.0.9-stable-bin.jar"/>
</runtime>
```

（3）<requires>项

```
<requires>
  <import plugin="org.eclipse.ui"/>
  <import plugin="org.eclipse.core.runtime"/>
</requires>
```

说明：在 requires 域中定义了该插件所要使用的依赖插件。现在两项就够了，随着开发的不断深入这里将会添加更多对其它插件的引用。如下是笔者的实际项目中的 requires 设置，它要用到 draw2d 和 gef 插件来画图、用于插件的帮助系统来创建建自己的帮助文档。

```
<requires>
    <import plugin="org.eclipse.ui"/>
    <import plugin="org.eclipse.core.runtime"/>
    <import plugin="org.eclipse.core.resources"/>
    <import plugin="org.eclipse.draw2d"/>
    <import plugin="org.eclipse.gef"/>
    <import plugin="org.eclipse.help"/>
    <import plugin="org.eclipse.help.ui"/>
    <import plugin="org.eclipse.help.appserver"/>
    <import plugin="org.eclipse.help.webapp"/>
</requires>
```

4、为 HelloWorld 修改 plugin.xml

将如下代码加入到 plugin.xml 的“</requires>”行之后：

```
<extension point="org.eclipse.ui.actionSets">
   <actionSet
         label="样本操作集"
         visible="true"
         id="myplugin2.actionSet">
      <menu
            label="样本菜单(&amp;amp;M)"
            id="sampleMenu">
         <separator
               name="sampleGroup">
         </separator>
      </menu>
      <action
            label="样本操作(&amp;amp;S)"
            icon="icons/sample.gif"
            class="book.chapter_5.SampleAction"
            tooltip="Hello，这是手工做的插件"
            menubarPath="sampleMenu/sampleGroup"
            toolbarPath="sampleGroup"
            id="book.chapter_5.SampleAction">
      </action>
   </actionSet>
</extension>
```

说明:

在<extension>项设置要扩展的扩展点，它是非常重要的一项。

point="org.eclipse.ui.actionSets"，设置了本插件的扩展点为何，actionSets 是指 Eclipse 的菜单、菜单项和工具栏按钮的扩展点

<actionSet>项表示一个 action 组(菜单、按钮)。label 是显示的名称。id 其唯一标识符，只要保证在本 plugin.xml 文件中不存在重复的 id 就行了。visible 指设置的按钮或菜单是否显示，如果设置成 false，则不显示。注意：要看 visible 设置的效果要将"透视图"关掉再重新打开。

<menu>是<actionSet>下的子项，它表示在 Eclipse 中插入显示一个名为"样本菜单(M)"的主菜单。separator 标签是一个结束符，它可以对菜单分组。

<action>也是<actionSet>下的子项，由它设置菜单、按钮。icon 是图片的路径，如果该图片不存，默认是一个红色实心小框（Eclipse2.X）或不显示图片而显示文字（Eclipse3.X）。Class 是按钮所对应的类，注意包名也要加上。menubarPath 表示把这个 action 做成一个菜单项放在上前<menu>定义的主菜单下。toolbarPath 表示把这个 action 再做成一个工具栏按钮。id 是标识符，设置成和 class 项一样的名称是个不错的选择。

以上仅是 Eclipse 的扩展点中的一种，此外还有其它的扩展点共有一百多种之多。我们没有必要了解所有扩展点的设置，只须熟悉一些常用的扩展点即可，如视图的扩展点 org.eclipse.ui.views、编辑器的扩展点 org.eclipse.ui.editors 等，本书将陆续给于介绍。另外，各种扩展点在 Eclipse 的帮助中有详细的说明，其位置为：选择主菜单"帮助→帮助内容"，然后打开"平台插件开发指南→参考→扩展点参考"项。

5、运行插件

按上一节（5.2.1 节）所说的方法运行插件（运行之前不妨将上节所建的 myplugin 项目关闭掉，关闭方法：右键单击 myplugin 项目名，然后在弹出菜单中选择"关闭项目"）。myplugin2 插件的效果如图 5.8 所示

图 5.8　myplugin2 插件运行效果图

5.3　常用插件扩展点实战（plugin.xml）

在上一节（5.2.2 节）已经对原有的 plugin.xml 做了很详尽的介绍，plugin.xml 是插件和 Eclipse 的接口，Eclipse 就象一所大宅子，它的外墙（plugin.xml）有很多的门（扩展点），我们要熟练进出这座大宅子，先得搞清楚它有哪些门，当然我们只需要熟悉一些主要的门就足够应付 90%的需求了。

本节将以开发需求为导向来介绍这些扩展点，并且本节所有实例都在 5.2.2 节所建立的 myplugin2 项目的基础上来进行讲解演示。

5.3.1 加入透视图（perspectives）

往开发一个插件，最常用的方式就是新增一个属于本插件专有的透视图，然后在此透视图基础上来展开软件开发，本书即采用这种方式。

1、准备工作

我们先将以前用到的那些图标的 icons 目录复制一份到 myplugin2 项目中，复制后的路径如图 5.9 所示：

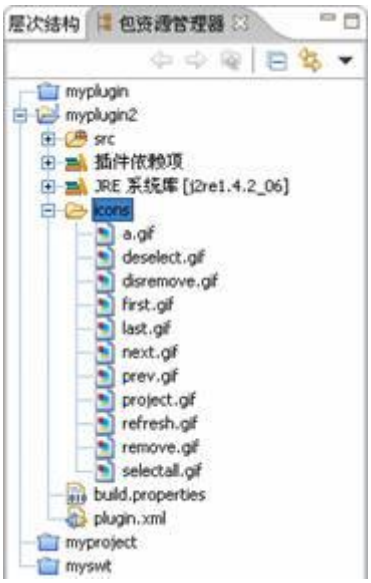


图 5.9 图标的路径

2、修改 plugin.xml 文件，设置透视图的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>之前：

```
<extension
     point="org.eclipse.ui.perspectives">
  <perspective
        name="myplugin 透视图"
        icon="icons/selectall.gif"
        class="book.chapter_5.SamplePerspective"
        id="book.chapter_5.SamplePerspective">
```

```
  </perspective>
</extension>
```

说明:

org.eclipse.ui.perspectives 是透视图的扩展点

name - 透视图的名称

icon - 透视图的图标

class - 透视图所对应的类（我们还没编写，下一步将完成此类）

id - 透视图标识，建议设置成和 class 一样的名称，省得以后扩展点设置得太多，搞得人糊涂。

3、建立透视图类

在上一步的 plugin.xml 中提前设置了透视图对应的类 book.chapter_5.SamplePerspective，这一步我们就来在包 book.chapter_5 中创建此类。透视图的类必须实现 IPerspectiveFactory 接口，此接口只有一个方法 createInitialLayout，我们让它先空实现好了。SamplePerspective 代码如下:

```
//--------文件名: SamplePerspective.java--------------------
public class SamplePerspective implements IPerspectiveFactory {
    public void createInitialLayout(IPageLayout layout) {}
}
```

4、运行插件

按以前所说的方法运行插件后，在新开的 Eclipse 环境中选择主菜单“窗口→打开透视图→其它”。在弹出如图 5.10 的透视图选择窗口中，我们可以看到一个名为“myplugin 透视图”的项。

图 5.10　选择透视图

选择并打开“myplugin 透视图”项后，显示如图 5.11 的 Eclipse 界面。我们发现该透视图光秃秃的什么也没有。没关系，我们下一小节就会往这个透视图加入两个视图。

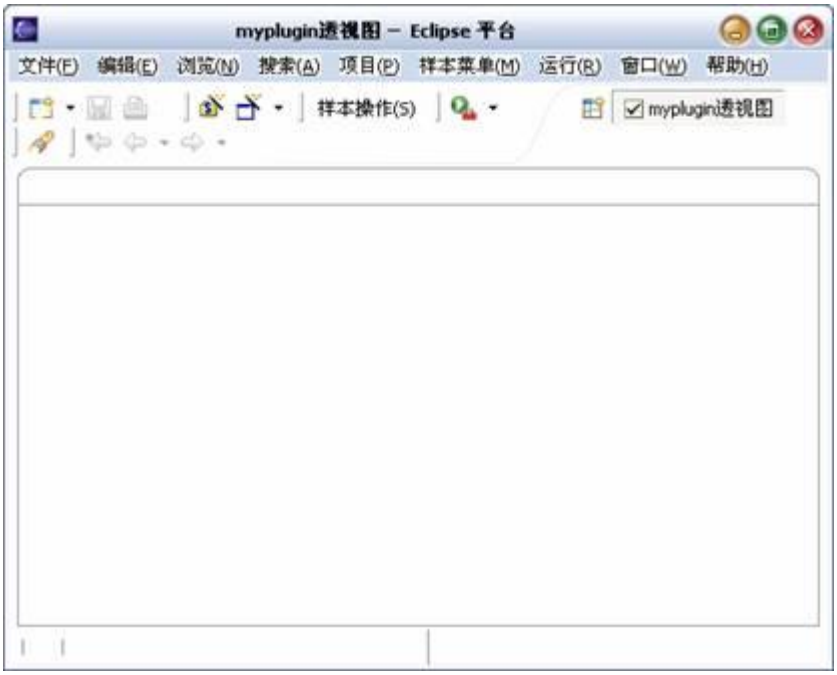


图 5.10　myplugin 透视图的效果图

5、总结

由本小节可以看到在 Eclipse 创建一个界面（菜单、按钮、透视图）是多么的简单，我们都不用编写实际界面的创建代码，只要设置一些扩展点就行了。

第 6 章　SWT 概述

在这一章里将把 SWT 和 AWT/SWING 做了简单的比较，并以一个 HelloWorld 的 Java 应用程序（Application）作为起步，让读者可以快速建立对 SWT/JFace 的感性认识。在这一章里所有的例子都是以 Java 应用程序方式来写的，之所以如此，是因为 Java 应用程序代码简洁，且可以独立运行，便于讲解和示范。当然，这些例子的代码方法同样适用于 Eclipse 的插件开发，SWT/JFace 在 Java 应用程序和 Eclipse 插件开发中的使用是没有太多区别的。

6.1　SWT 简介

2003 年，笔者对 SWT/JFace（英文全称：Standard Widget Toolkit）还是仅有耳闻，知道在 AWT/Swing 之外，又有了一个新的图形 API 包，听说还很不错，当时书店里根本没有相关资料，只能在网上找到一些零星的文章来了解。

2004 年前，笔者还极少用 Java 来写 GUI 程序（GUI 全称：Graphical User Interfaces，图形用户界面），主要的工作都是用 JSP 来写网页。用 JAVA 来开发大型的 GUI 程序实在

很困难的事，大都丑陋又笨重（慢），SUN 在 GUI 方向上的失败是公认的事实。失败关键之处在于 Java 的图形 API 包 AWT/SWING 在速度和外观上都不能让人满意，外观总是和同操作系统平台下的其他软件格格不入，对机器配置的需求也似乎永无止境。

2004 年初，笔者有幸参与到一个用 Eclipse 插件方式来开发的软件项目中，该软件使用到了 SWT/JFace，那界面实在是太酷太漂亮了，让人为之耳目一新，而且界面响应速度极快，这真的是用 Java 开发的吗？当时竟然有点不敢相信。

无疑，SWT/JFace 象一股清新的风吹入了 Java 的 GUI 开发领域，为这个沉闷的领域带来了勃勃生机。虽然 SUN 不接纳 SWT/JFace 作为 Java 中的一种图形 API 标准，但它虽然借着 Eclipse 的优异表现，以不可阻挡之势向前发展着。终于可以用 SWT 轻松的开发出高效率的 GUI 程序，且拥有标准的 Windows 外观，Eclipse 软件就是基于 SWT/JFace 构建的，大家看看 Eclipse3.0 就知道 SWT 有多么的棒。

图 6.1　SWT、JFace、GUI 程序三者关系示意图

如上图 6.1，为了方便开发 SWT 程序，在 SWT 基础上又创建了一个更易用、功能强大的图形包"JFace"。然而，JFace 并不能完全覆盖 SWT 的所有功能，所以编程时 SWT、JFace 都会要用到，但是一般来说，能用 JFace 的组件就最好不要用 SWT 的。

6.2　SWT 中的包

SWT 是 Eclipse 图形 API 的基础，本节将简单介绍一下 SWT 中所包含的子包。

1、org.eclipse.swt.widgets

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

2、org.eclipse.swt.layout

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

3、org.eclipse.swt.custom

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

4、org.eclipse.swt.event

SWT 采用了和 AWT/SWING 一样的事件模型，在包中可以找到事件监听类和相应的事件对象。比如，鼠标事件监听器 MouseListener，MouseMoveListener 等，及对应的事件对象 MouseEvent。

5、org.eclipse.swt.graphics

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

6、org.eclipse.swt.ole.win32

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

此外还有 org.eclipse.swt.dnd、org.eclipse.swt.printing、org.eclipse.swt.program、org.eclipse.swt.accessibility、org.eclipse.swt.browser、org.eclipse.swt.awt 等包，在此不一一介绍了。这些包一般很少用到，只需要稍微了解一下就行了，不必深究。

6.3 用 SWT Designer 写一个 Hello World

SWT Designer 是优秀的 SWT/JFace 开发辅助工具，本书大都 SWT/JFace 的例子都是使用它来生成代码后，再进行修改而成。当然，SWT Designer 并非是阅读和运行这些例子的必须条件。

本节将用 SWT Designer 来写出第一个基于 SWT 的 HelloWorld 程序，以此给读者演示在开发中是如何使用 SWT Designer 的。

6.3.1 使用用向导建立一个 SWT/JFace Java 项目

（1）选择主菜单“文件→新建→项目”，弹出如下图 6.2 所示窗口。

图 6.2 新建项目窗口

（2）选择“Designer”下的“SWT/JFace Java Project”项，单击“下一步”，弹出如下图 6.3 所示窗口。

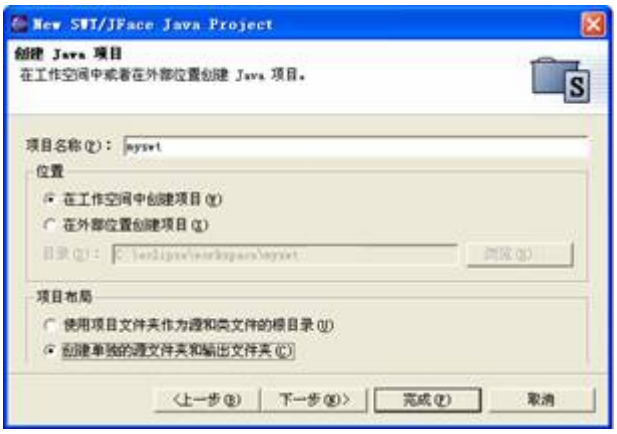


图 6.3　创建 Java 项目窗口

（3）填写项目名称“myswt”，项目布局选择第二个，单击“完成”。这时如果打开“java”透视图，可以看到多了一个名为“myswt”的项目，下方还排列着很多库引用，如下图 6.4 所示窗口。

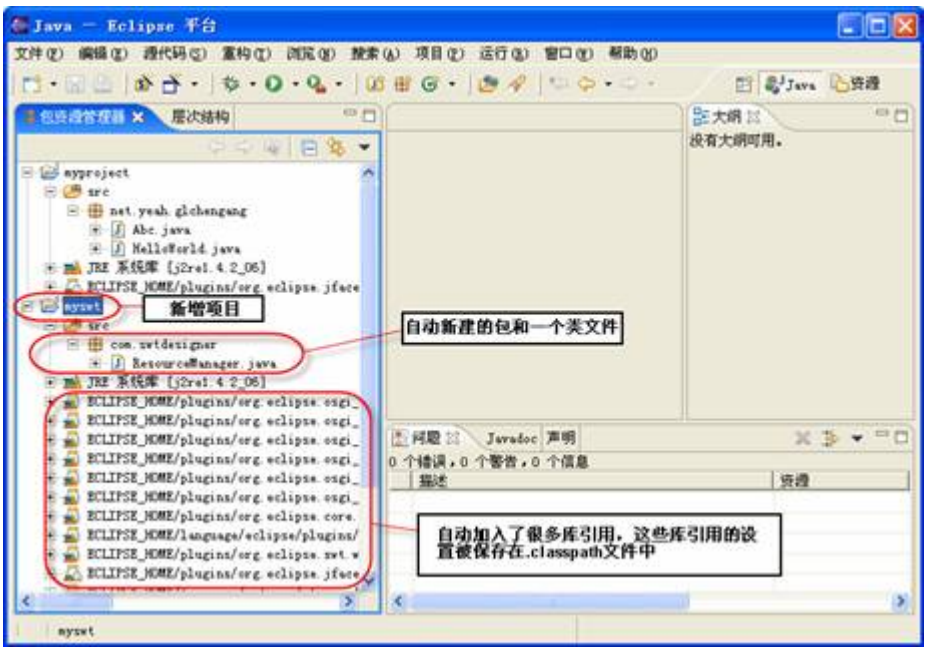


图 6.4　“java”透视图

注：

（1）其实写 SWT 程序也不是一定要重新建立这样一个新的项目，原来老的“myproject”项目依然可以继续使用的，但必须将 SWT、JFace 包及一些相关的包引用到 Java 构建路径中，手工一步步做这个工作太过于繁锁。有一个简单的方法：借助 SWT Designer 新建项目时保存在.classpath 文件中的库引用，将其复制粘贴到 myproject 的.classpath 中即可。

（2）当编写 Java 程序时，笔者认为“Java”透视图要比默认的“资源”透视图好用，主要是因为前者的包显示不是树状的，用起来较方便。但选择哪一种透视图，还是要看各人的习惯和喜好。本书以后的所讲内容将统一使用“Java”透视图。

6.3.2　导入 SWT 的原生库

想要运行 Java 应用程序，必须将 SWT 的原生包导入到项目中，否则该项目在运行程序时会报异常“java.lang.UnsatisfiedLinkError: no swt-win32-3063 in java.library.path”，并弹出图 6.5 所示的错误提示框。

图 6.5 未导入 SWT 原生包时产生的错误提示框

导入 SWT 原生包的步骤如下:

（1）右键单击项目名“myswt”，在弹出菜单中选择“导入”，则会弹出如图 6.6 所示窗口。

图 6.6 导入窗口

（2）选择“文件系统”后单击“下一步”，转到如图 6.7 所示窗口

图 6.7　选择导入文件

（3）通过“浏览”按钮找到 SWT 原生库的路径（也可以直接输入路径文字），路径为“C:\eclipse\plugins\org.eclipse.swt.win32_3.0.1\os\win32\x86”。然后将“swt-win32-3063.dll”选上，单击“完成”，导入 SWT 原生包的设置结束。

6.3.3　新建一个 SWT 类文件

参阅“4.2 节 创建 Java 项目并运行”所讲方法，新建一个类文件。

（1）在“Java”透视图的“包资源管理器”中，右键单击“com.swtdesigner”包，在弹出菜单中选择“新建→其他”，弹出如图 6.8 所示窗口。

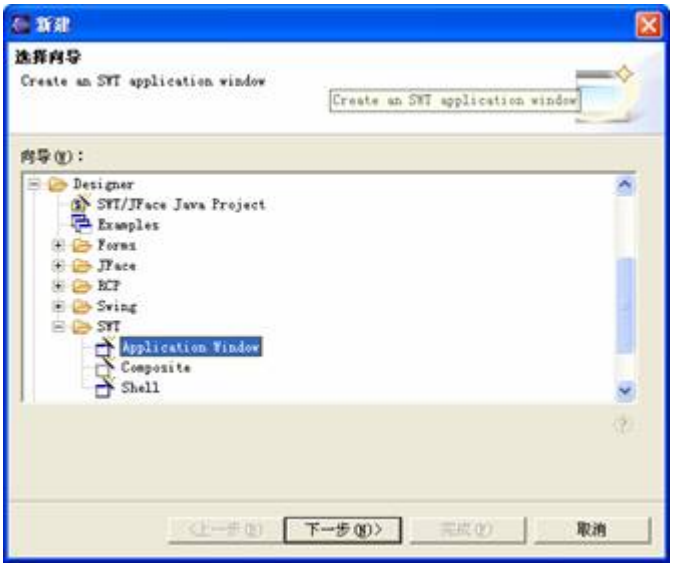


图 6.8　选择新建的类型

（2）选择“Designer→SWT→Application Window”，单击“下一步”，弹出如图 6.9 所示窗口。

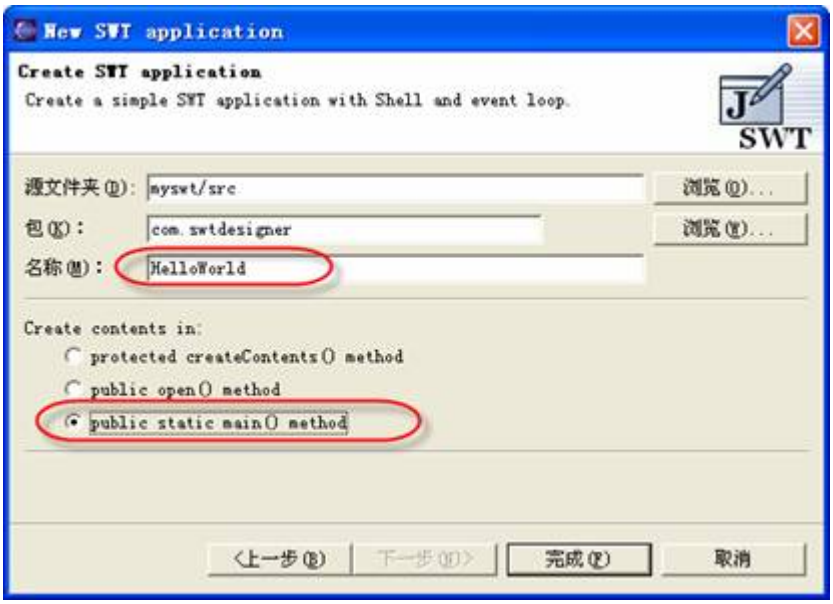


图 6.9　类文件的设置

（3）类的名称填“HelloWorld”，并选择“Create contents in（类代码的生成方式）”为第三项“public static main() method”（第三项生成的代码结构最简单），弹击“完成”。Eclipse 将自动生成 HelloWorld.java 的代码，代码如下（注释为笔者手工加入）：

```
package com.swtdesigner; //包名

import org.eclipse.swt.widgets.Display;//程序所用到的类都会用 import 标记在这里，

import org.eclipse.swt.widgets.Shell;     //import 的快捷键 Ctrl+Shift+O

public class HelloWorld {  //一个标准的 Java 类 HelloWorld

    public static void main(String[] args) {

            //display 负责管理事件循环和控制 UI 线程和其他线程之间的通讯。

        final Display display = Display.getDefault();

        final Shell shell = new Shell();  // shell 是程序的主窗口

        shell.setSize(327, 253);  //设置主窗口的大小

        shell.setText("SWT Application");  //设置主窗口的标题

        shell.layout();  //shell 应用界面布置

        shell.open();  //打开 shell 主窗口

        while (!shell.isDisposed()) { //如果主窗口没有关闭，则一直循环

            if (!display.readAndDispatch())  //如果 display 不忙

                display.sleep(); //display 休眠

        }

    }

}
```

从这个代码可以看到，创建一个典型的 SWT 应用程序需要以下步骤：

创建一个 Display

创建一个或多个 Shell

设置 Shell 的布局（3.5 节将讲到布局的内容）

创建 Shell 中的的组件（注：本例还没有加入组件，只是一个空窗口）

用 open()方法打开 Shell 窗口

写一个事件转发循环

销毁 display

6.3.4 在主窗口加入一个文本框组件

如果运行 HelloWorld.java，它还仅是一个空荡荡的主窗口。我们利用 SWT Designer 将一个 SWT 的文本框组件加入到主窗口中，操作步骤如图 6.10 所示。

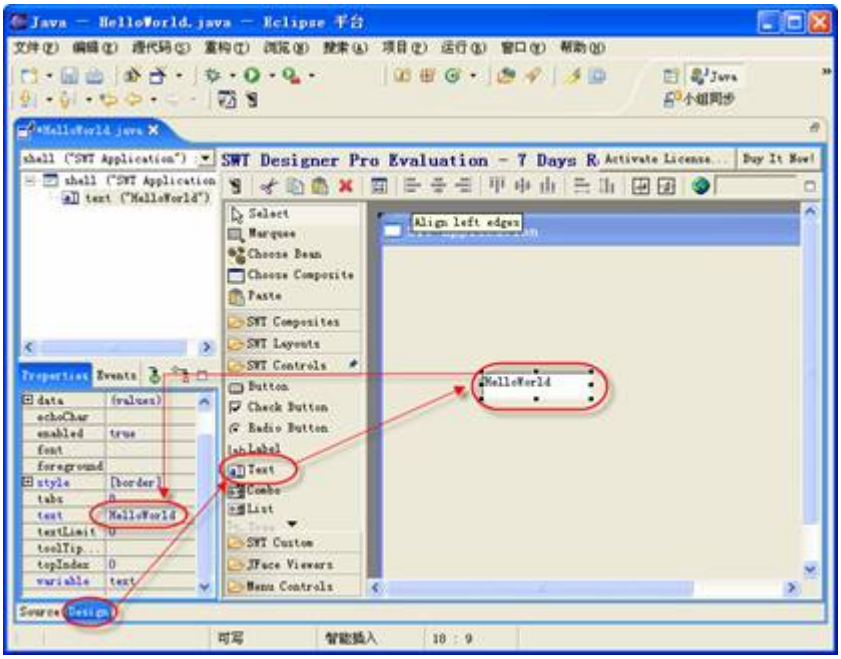


图 6.10 将文本框加入到主窗口的操作示意图

图中所示的操作步骤用文字描述如下:

（1）先将编辑器最大化。然后单击 Eclipse 的左下角的“Design”选项页，则编辑器由代码视图变成设计视图。

（2）选择 SWT 组件栏中“SWT Controls”分类下的“Text”组件，然后在主窗口上单击，将 Text 框放入。注意这里不是通常的将组件拖入到窗口。

（3）转到属性窗口，在“text”项里填写“HelloWorld”。单击 Eclipse 左下角的“Source”返回到编辑器的代码视图，代码如下:

```
package com.swtdesigner;

import org.eclipse.swt.widgets.Display;

import org.eclipse.swt.widgets.Shell;

import org.eclipse.swt.SWT;

import org.eclipse.swt.widgets.Text;

public class HelloWorld {
```

```
    public static void main(String[] args) {
        final Display display = Display.getDefault();
        final Shell shell = new Shell();
        shell.setSize(327, 253);
        shell.setText("SWT Application");
        //------------------新插入的界面核心代码
-------------------------
        Text text = new Text(shell, SWT.BORDER); //新建一个 text 对象
         text.setText("HelloWorld"); //给 text 文本框设置初始文字
HelloWorld
        text.setBounds(88, 94, 100, 25); //设置文本框的位置和大小，(x
轴坐标,y 轴坐标,宽度,高度)
        //------------------END----------------------------------
----------
        shell.layout();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
    }
}
```

6.3.5 运行 HelloWorld.java

选择主菜单“运行→运行方式→Java 应用程序”，运行界面如图 6.11 所示:

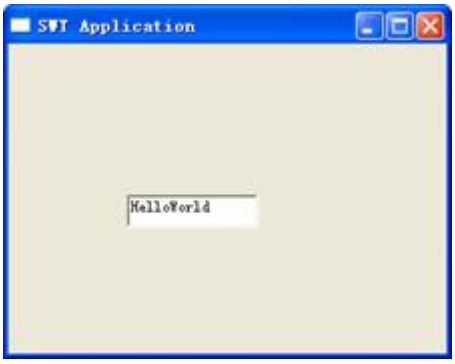


图 6.11 HelloWorld 的第一次运行界面

以上的程序例子还是比较简单的，如图 6.12 所示，给出一个稍微标准些的界面，并给出了各类和界面之间的对应关系。注：在 SWT 中 check 框（复选框）也是一种 Button。

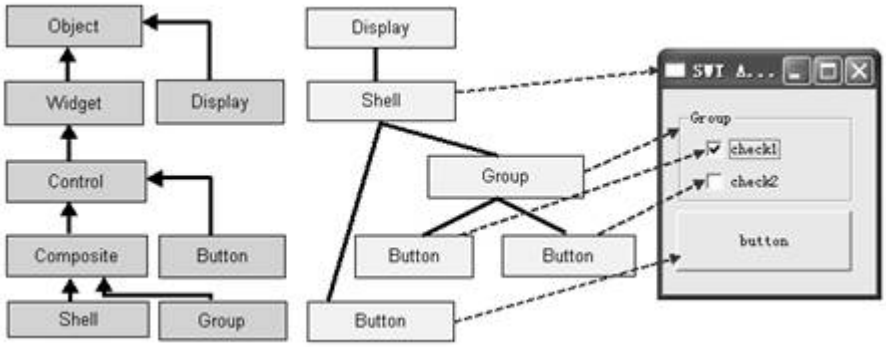


图 6.12　界面和类之间的对应关系图

其中 Display 和 Shell 的谱系图如图 6.13 所示，Group 和 Button 在 3.3 节有介绍。

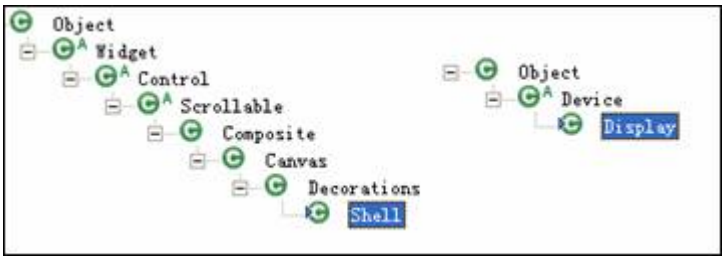


图 6.13　Display 和 Shell 的谱系图

6.4　关于 SWT/JFace 例程的说明

由于 SWT/JFace 应用程序例子的整体代码结构都基本一样，如下：

```
package com.swtdesigner;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Text;
public class HelloWorld {
    public static void main(String[] args) {
        final Display display = Display.getDefault();
        final Shell shell = new Shell();
        shell.setSize(327, 253);
        shell.setText("SWT Application");
        //-----------------新插入的界面核心代码----------
```

```
            …… ……
            //----------------END-----------------------
        shell.layout();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
    }
}
```

为了节省篇幅，以后的例子一般都会省略上面代码框架前后部份，只给出中间省略号处的核心代码，要想得到完整的代码请查阅本书随书光盘中的例程。

6.5 实践建议

SWT Designer 还无法完成所有的界面设计工作，所以在界面开发中依然是以手工写代码为主，而且手写代码某些时候比界面拖拉操作更快捷。以下是笔者在使用 SWT Designer 开发界面时的基本流程:

新开一个临时的 Application 文件，用 SWT Desiger 快速做好开发所需要的部份界面。

将自动生成的代码移植到正式项目中，进行手工修改和代码精简。

另外，由于 SWT Designer 不是很稳定，所以在使用时还应注意:

不要在界面中加入太多组件。

不要频繁的移动组件，或者删除又添加组件，否则很可能因为内存耗尽而死机。

6.6 本章小结

本章主要介绍了 SWT 的一些基本知识，并且用 SWT Designer 开发出了本书的第一个 SWT 程序。通过这章的学习，读者对 SWT 有一个初步的认识，并了解到了如何用 SWT Designer 来开发 SWT 程序。

第 7 章 SWT/JFace 的事件模型

7.1 事件的四种写法

SWT 的事件模型是和 Java 标准的 AWT 基本一样的。在第 6 章的例子中，如何来实现文本框的事件响应呢？比如：鼠标双击文本框弹出一个对话框。下面将按照事件的四种写法来实现它。

7.1.1 匿名内部类写法

在原来的代码行“text = new Text(shell, SWT.BORDER);”之下插入如下语句：

```
//addMouseListener 加入鼠标事件的监听器
text.addMouseListener(new MouseAdapter() {
    public void mouseDoubleClick(MouseEvent e) {//鼠标双击事件的方法
         //打开一个信息框
        MessageDialog.openInformation (null,"","Hello World");
    }
});
```

new MouseAdapter()就是一个匿名内部类。我们建立了一个继承于 MouseAdapter 的类，但并没有给这个类命名，并且没有用通常的写法，而是直接在 text.addMouseListener 方法中写下了类的代码，这就是所谓的匿名内部类（更详尽的解释请参阅 Java 基础类书籍）。

使用匿名内部类来写事件代码简单方便，但也要注意它的一些缺点：

由于事件处理代码会随着组件一起分散在代码中的各个部份，不够集中，这样会导致代码阅读与维护上的不便。

各事件的处理全部由嵌套的程序块组成，视觉上会显示有些乱。如果事件处理代码很长，也会导致了阅读与维护上的不便。

当工具栏、菜单栏目等也需要处理相同的用户行为时，无法重用事件中的处理代码，导致了代码的臃肿。

7.1.2 命名内部类写法

事件代码使用命名内部类的方式，可以解决匿名内部类存在的问题：首先，事件处理代码都集中在一起，并且都具有有意义的名称，程序容易阅读与维护；另外，单个的事件处理程序也可以被工具栏、菜单栏等重用。实现代码如下：

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

```
        Text text = new Text(shell, SWT.BORDER);
          //加入鼠标事件监听器，并用下面代码所定义的内部类生成一个对象
        text.addMouseListener(new MyMouseDoubleClick());
        ……
    }

      //定义一个名为 MyMouseDoubleClick 的内部类
    private static final class MyMouseDoubleClick extends
MouseAdapter {
        public void mouseDoubleClick(MouseEvent e) {
            MessageDialog.openInformation(null, "", "Hello World");
        }
    }
}
```

7.1.3 外部类写法

这种写法和命名内部类有些相似，只不过是将 MyMouseDoubleClick 类从 HelloWorld.java 中拿出去，单独写成一个类文件。这种写法有和命名内部类一样的优点，但因为要单独写成一个文件，写起来会麻烦一些。实现代码如下

```
//文件 1:  HelloWorld.java
public class HelloWorld {
    public static void main(String[] args) {
            ……
        Text text = new Text(shell, SWT.BORDER);
          //加入鼠标事件监听器，并用下面代码所定义的内部类生成一个对象
        text.addMouseListener(new MyMouseDoubleClick());
        ……
    }
}
```

```
//文件 2: MyMouseDoubleClick.java
public class MyMouseDoubleClick extends MouseAdapter {
    public void mouseDoubleClick(MouseEvent e) {
        MessageDialog.openInformation(null, "", "Hello World");
    }
}
```

7.1.4 实现监听接口的写法

将 HelloWorld 类实现 MouseListener 接口，这样类本身就成了一个监听器，使得加入监听器的代码可以更简洁。这种方式适合加入监听器的组件较多，且要求监听器的事件处理代码可以被组件共用。这种方式还有一个要注意的地方：事件方法和其他方法混合在了一起，容易引起误读，所以应该在事件方法前加入详细的注释说明。

实现 MouseListener 接口要写的事件方法多一些，当然没用的事件方法可以空实现。如果继承 MouseListener 接口的适配器 MouseAdapter，则只写需要的方法就行了。另外要注意：只有接口才能有多继承的特性，所以如果 HelloWorld 已经是某个类的子类，就只能用实现接口的方式，而不能继承接口的适配器了。

给出示例代码如下：

```
public class HelloWorld extends MouseAdapter{//或 implements
MouseListener
    public static void main(String[] args) {
            ……
        Text text1 = new Text(shell, SWT.BORDER);
        Text text2 = new Text(shell, SWT.BORDER);
        text1.addMouseListener(this);
        text2.addMouseListener(this);
        ……
    }

    public void mouseDoubleClick(MouseEvent e) {
```

```
        MessageDialog.openInformation(null, "", "Hello World");
    }
}
```

7.1.5 总结

匿名内部类方式在写起来方便些，但不适合事件代码太长太多的情况。从代码书写、阅读、维护以及程序的可扩展性角度来看，命名内部类写法最为值得推荐。外部类的写法主要是为了代码重用才考虑使用，如果包（package）外的类要用到此事件处理代码，这时外部类就派上用场了。而第四种写法，要求组件都可以共用事件代码时才能使用。

7.2 常用事件介绍

除了上例中用于响应鼠标事件的 addMouseListener，Eclipse 还有一些常用的监听器，它们在各组件中的使用方法相同（如果该组件支持此种事件的话）。在此将它们简单介绍如下：

addSelectionListener：这个监听器最最常用。

a) widgetSelected 方法：当组件被选择（鼠标单击、按回车键）时触发此方法的事件处理程序。

b) widgetDefaultSelected 方法：用于某些很少触发选择事件的组件，所以这个方法在实际开发中也很少用。比如，文本框回车事件、列表框双击事件等，就只能用 widgetDefaultSelected 方法，用 widgetSelected 方法无效。

addKeyListener（按键）

a) keyPressed 方法：当前焦点停在组件时，按下键盘任一键时触发。但对于某些组件（如按钮 Button）按回车键无法执行此方法。

b) keyReleased 方法：按键弹起时触发。

addFocusListener（焦点）

a) focusGained 方法：得到焦点时触发。

b) focusLost 方法：失去焦点时触发

addMouseListener（鼠标）

a) mouseDown 方法：鼠标按下时触发

b) mouseUp 方法：鼠标放开时触发

c) mouseDoubleClick 方法：鼠标双击时触发

以上几个就是常用的事件了，很少吧，SWT 的事件模型还是极容易掌握的。事实上除了 addSelectionListener 较常用之外，其他基本都很少用到。

7.3 在事件代码中如何访问类中的变量

7.3.1 访问类中的变量的三种方法

在写事件代码的时候，常常需要引用主类中的变量，要访问这些变量是需要一些技巧的。

方法一：加 final 修饰符。

```
public class HelloWorld {
    public static void main(String[] args) {
          ……
          //将变量前加 final，否则在事件代码里不能引用
          final String str="陈刚";
          text.addMouseListener(new MouseAdapter() {
            public void mouseDoubleClick(MouseEvent e) {
              System.out.println(str); //str 变量前要加 final
            }
          });
          ……
    }
}
```

方法二：将变量 str 变成类的实例变量。但这种方法扩大了 str 变量的有效范围。

```
public class HelloWorld {
    //由于引用它的代码是在静态方法内才加 static，否则不必要 static。
       static String str="陈刚";
    public static void main(String[] args) {
          ……
    }
}
```

方法三：将事件代码写成命名内部类，然后通过构造函数的参数来传入。这种方法较繁琐一些。

```
public class HelloWorld {

    public static void main(String[] args) {

        String str="陈刚";

        //通过构造函数参数将 str 值传入

        text.addMouseListener(new MyMouseDoubleClick(str));

    }


   //匿名内部类 MyMouseDoubleClick

    private static final class MyMouseDoubleClick extends
MouseAdapter {

        private String string;//建一变量引用 str 的值

        public MyMouseDoubleClick(String str){ //通过构造函数参数接
受 str 值

            this.string=str;

        }

        public void mouseDoubleClick(MouseEvent e) {

            System.out.println(string);

        }

    }

}
```

7.3.2 Java 中变量的称法和说明

此节中用到了一些 Java 变量方面的知识，在此一并附上。Java 中有三种容易混淆的变量：局部变量、实例变量、类变量，如下程序所示：

```
public class Variable {

    static int allClicks = 0; //类变量

    String str = "广西桂林"; //实例变量
```

```
    public void method() {
        int i = 0; //局部变量
    }
}
```

类变量的定义前加有 static，这表示它是一个静态的，因此类的多个实例共用一个类变量。实例变量定义在类的方法之外，一般处于类的起始位置，类的每一个实例都独自拥有一份实例变量的拷贝。局部变量的有效范围在程序块中，它的生命期仅限于此程序块内。

实例变量在有些书籍中也翻译成“域”或“成员变量”。在面向数据库的实体类（Hibernate 中也称 POJO – 简单原始的 Java 对象）中，被称之为“属性”或“字段”的变量，也是实例变量的一种。

使用变量的一般原则是，尽量使变量的有效范围最小化：优先考虑用局部变量，其次是实例变量，最后才是类变量。

另外，还有一种常量的写法，它比类变量写法仅多了个 final，如下所示：

```
final static int ALL_CLICKS = 0; //常量
```

注意 ALL_CLICKS 是全大写的，这是常量的规范命名方式。这时 ALL_CLICKS 被 final 约束，它不能再被赋值了。

7.4 本章小结

本章主要介绍了事件的四种写法，及事件访问类中变量的方法，这些知识在 SWT 编程中经常要用到。但是，读者可以不必太执着于本章的内容，可以快速浏览后，进入下一章的学习，当用到时，再回过头来查阅。

第 18 章 常用插件扩展点

在第 17 章对 plugin.xml 作了少量介绍，plugin.xml 是插件和 Eclipse 内核的接口，Eclipse 就像一所大宅子，它的外墙（plugin.xml）有很多的门（扩展点），要熟练进出这座大宅子，先得搞清楚它有哪些门（扩展点）。
插件的扩展点非常之多，但很多扩展点都不常用到，只要熟悉一些主要的扩展点即可。本节将面向实际开发需要来介绍这些扩展点，并且本章所有实例都在第 17 章建立的 myplugin2 插件项目的基础上创建。

18.1 加入透视图（perspectives）

开发一个插件，最常用的方式就是新增一个属于本插件专有的透视图，然后在此透视图基础上来展开功能，本书也采用这种方式。

18.1.1 准备工作

先将以前用到的包括图标的 icons 目录复制一份到 myplugin2 项目中，复制后的路径如图 18.1 所示。

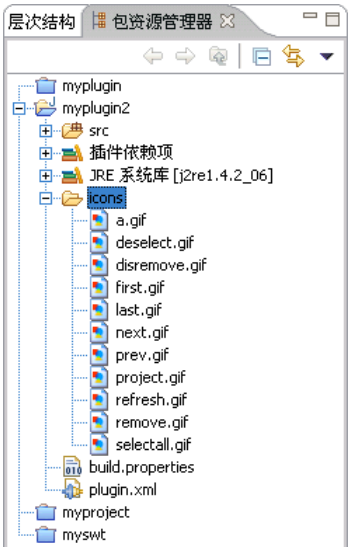


图 18.1　图标的路径

18.1.2 修改plugin.xml文件，设置透视图的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>项之前:

```
<extension
    point="org.eclipse.ui.perspectives">
  <perspective
       name="myplugin 透视图"
       icon="icons/selectall.gif"
       class="cn.com.chengang.SamplePerspective"
       id="cn.com.chengang.SamplePerspective">
  </perspective>
</extension>
```

代码说明:

- org.eclipse.ui.perspectives 是透视图的扩展点。
- name: 透视图的名称。
- icon: 透视图的图标。
- class: 透视图所对应的类（还没编写，下一步将完成此类）。
- id: 透视图标识，建议设置成和 class 一样的名称，省得以后扩展点设置得太多，让人糊涂。

18.1.3 建立透视图类

在 18.1.2 小节的 plugin.xml 中提前设置了透视图对应的类 cn.com.chengang.Sample Perspective，这一步就在包 cn.com.chengang 中创建此类。透视图类必须实现 Iperspective Factory 接口，此接口只有一种方法 createInitialLayout，先让它空实现。
SamplePerspective 类的代码如下:

```
//--------文件名: SamplePerspective.java--------------------
public class SamplePerspective implements IPerspectiveFactory {
   public void createInitialLayout(IPageLayout layout) {}
}
```

18.1.4 运行插件

运行插件，然后在新 Eclipse 环境中选择主菜单"窗口→打开透视图→其他"选项。在弹出窗口中，可以看到一个名为 myplugin 透视图的项，如图 18.2 所示。
选择并打开"myplugin 透视图"选项后，显示如图 18.3 所示的 Eclipse 界面。我们发现该透视图光秃秃的什么也没有。没关系，下面就会向这个透视图中加入两个视图。

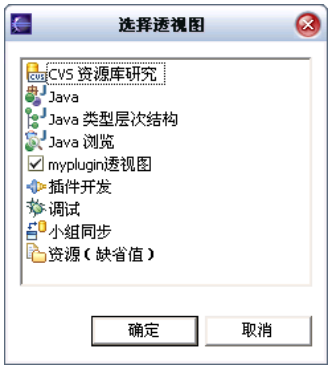


图 18.2　选择透视图

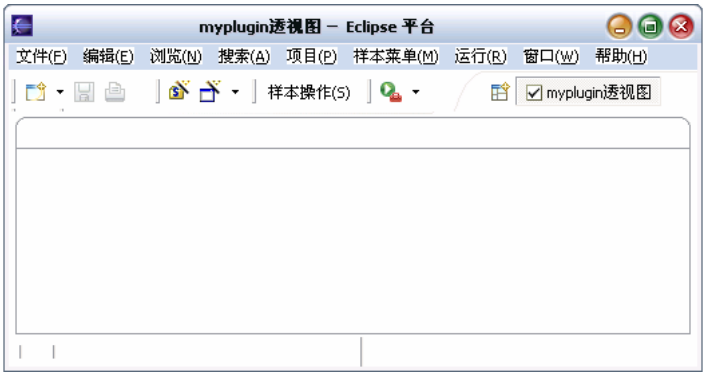


图 18.3　myplugin 透视图的效果图

18.1.5　总结

由本节可以看到，在 Eclipse 插件环境中，创建一个菜单、按钮、透视图界面是多么简单，都不用编写实际界面的创建代码，只要设置一些扩展点就行了。

18.2　在透视图中加入视图（views）

接着 18.1 节的内容，给透视图加入两个视图，实现的步骤如下所述。

18.2.1　修改plugin.xml文件，设置视图的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>之前：

```
<extension
    point="org.eclipse.ui.views">
  <category
        name="myplugin2 视图"
        id="com.glchengang.myplugin2.view">
  </category>
  <view
        name="视图 1"
        icon="icons/prev.gif"
        category="com.glchengang.myplugin2.view"
        class="cn.com.chengang.View1"
        id="cn.com.chengang.View1">
  </view>
```

```
  <view
        name="视图 2"
        icon="icons/project.gif"
        category="com.glchengang.myplugin2.view"
        class="cn.com.chengang.View2"
        id="cn.com.chengang.View2">
  </view>
</extension>
```

代码说明:

- org.eclipse.ui.views 是视图的扩展点。
- <category>…</category>是视图的分组名及 id 标识，它的效果体现在"显示视图"窗口里，显示视图的打开方法是: 在主菜单选择"窗口→显示视图→其他"选项，图 18.4 中的左图是本例设置的效果。
  <category>项中的 id 属性要保证它在 Eclipse 的所有插件中惟一。如果和 Ant 插件的 id 相同，原 Ant 组就会被 myplugin2 视图组抹掉了（右图）。如果删除掉<category>不设置，则 Eclipse 会自动新增一个"其他"组，并将两视图加入（中图）。
- <view>的 category 是表示本视图属于哪个组，与上面<category>项的 id 值相同。
- <view>的 class 是视图所对应的类（还没编写，下一步将完成这两个类）。
- <view>的 id 是视图标识，建议设置成和 class 一样的名称。

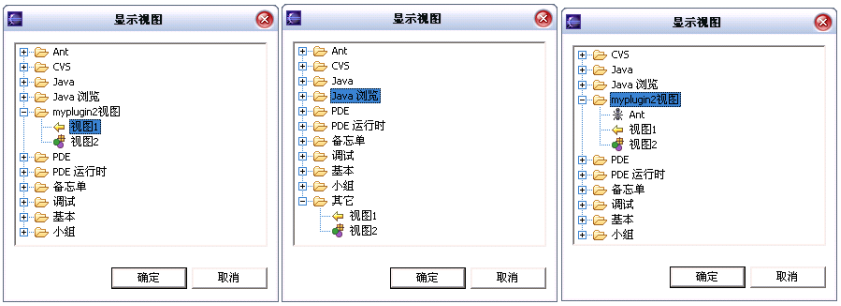


图 18.4　显示视图窗口

18.2.2　创建视图类

在 18.2.1 小节的 plugin.xml 中提前设置了视图对应的类：cn.com.chengang.View1、View2，本小节就来在包 cn.com.chengang 中创建这两个视图类。

视图的类必须继承抽象类 ViewPart，此类有两种方法 createPartControl、setFocus。我们要在 createPartControl 方法创建两个视图的界面组件，第一个视图创建一个列表，第二个视图创建一个文本框。而 setFocus 是关于视图焦点的方法，一般都不用写，让它空实现。

两类的代码如下:

```
//--------文件名: View1.java--------------------
public class View1 extends ViewPart {
    public void createPartControl(Composite parent) {
```

```
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        List list = new List(topComp, SWT.BORDER);
        list.add("中国");
        list.add("美国");
        list.add("法国");
    }
    public void setFocus() {}
}
```

```
//--------文件名: View2.java--------------------
public class View2 extends ViewPart {
    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        Text text = new Text(topComp, SWT.BORDER);
        text.setText("我是 text 框");
    }
    public void setFocus() {}
}
```

18.2.3 修改透视图类SamplePerspective

在 18.1 节加入一个透视图时，建立了一个透视图类 SamplePerspective，当时有一个继承自接口的方法 createInitialLayout，它还是空实现的。本例将通过修改这种方法，将两个视图加入到透视图中。createInitialLayout 方法的代码如下:

```
//参数 IPageLayout 是用于透视图的布局
public void createInitialLayout(IPageLayout layout) {
    //得本透视图的编辑空间标识
    String editorArea = layout.getEditorArea();
    /*
     * 将视图 1 加入到透视图的左部
     * "left" 视图区的 id 标识为"left"
     * IPageLayout.LEFT 在透视图布局中的位置靠左
     * 0.2f 占用透视图 20%的宽度
     * editorArea 使用透视图的编辑空间
     */
    IFolderLayout     left     =     layout.createFolder("left",
IPageLayout.LEFT, 0.2f, editorArea);
    left.addView("cn.com.chengang.View1"); //参数为 plugin.xml 中
视图 1 的 id 标识
    //将视图 2 加入到透视图的底部
    IFolderLayout    bottom    =    layout.createFolder("bottom",
IPageLayout.BOTTOM, 0.8f, editorArea);
    bottom.addView("cn.com.chengang.View2");
    /*
     * 将 17.2.2 小节第 4 点所定义的 actionset（主菜单、工具栏按钮）加入到
本透视图。这个效果要在
     * plugin.xml 文件的 action 设置中将 visible="false"才看得出效果，
这时打
```

```
     * 开其他透视图，action 设置的主菜单、工具栏按钮将不会出现在界面上，只有打
     * 开本透视图才会出现，因为本透视图用下面的语句手工加入了此 action
     * 参数 myplugin2.actionSet 为 action 在 plugin.xml 文件中设置的 id 标识
     */
    layout.addActionSet("myplugin2.actionSet");
}
```

18.2.4 运行插件

运行插件后，打开 myplugin 透视图，效果如图 18.5 所示。如果两个视图还没有显示在透视图上，则需要把透视图先关闭，再打开，以应用新的透视图设置。

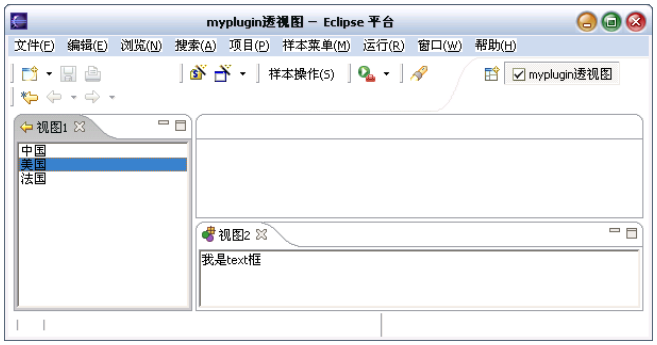


图 18.5 加入两视图后的透视图

18.3 在视图之间实现事件监听

两个视图中的组件之间的互动，在开发插件时是经常碰到的问题。例如，在 18.2.4 小节的界面中，单击视图 1 列表中的某项时，视图 2 的文本框也作相应显示。本节将实现此功能。

18.3.1 修改View1.java、View2.java

首先，要在两视图中互动就必须先解决“如何在视图 1 中取得视图 2 的对象”的问题。Eclipse 通过 plugin.xml 来加载插件和插件中的扩展点（如视图），所以可以由视图的 id 标识来取得视图对象，具体语句如下：

```
IWorkbenchPage wbp = getViewSite().getPage();
IViewPart view2 = wbp.findView("cn.com.chengang.View2");
```

得到了视图 2 的对象后，其他一切就都好办了，先给出修改后 View1 如下：

```
public class View1 extends ViewPart {
    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        final List list = new List(topComp, SWT.BORDER);
        list.add("中国");
        list.add("美国");
        list.add("法国");
        //列表选择事件监听
        list.addSelectionListener(new SelectionListener() {
            public void widgetSelected(SelectionEvent e) {
                //由 IWorkbenchPage 取出 view2
                IWorkbenchPage wbp = getViewSite().getPage();
                IViewPart                    view2                    =
wbp.findView("cn.com.chengang.View2");
                //将当前选择的列表项显示在文本框中
                Text text = ((View2) view2).getText();
                text.setText(list.getSelection()[0]);
            }
            public void widgetDefaultSelected(SelectionEvent e) {}
        });
    }
    public void setFocus() {}
}
```

然后将 View2 的文本框对象改成类的实例变量，并编写它相应的 set/get 方法，代码如下：

```
public class View2 extends ViewPart {
    private Text text;
    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        text = new Text(topComp, SWT.BORDER);
        text.setText("我是 text 框");
    }
    public void setFocus() {}
    //文本框 text 相应的 set/get 方法
    public Text getText() {return text;}
    public void setText(Text text) {this.text = text;}
```

```
}
```

18.3.2 总结

（1）在插件中 IWorkbenchPage 对象比较重要，在这里再给出一种获得此对象的方法。当不是在视图里而是在 Action 里要取 IWorkbenchPage 对象时，就可以用下面的方法：

```
Myplugin2Plugin.getDefault().getWorkbench().getActiveWorkbench
Window(). getActivePage();
```

（2）IWorkbenchPage.findView("cn.com.chengang.View2")，中的参数为“视图2”在 plugin.xml 中设置的 id 标识，由此可见 plugin.xml 文件在插件中的地位是极其重要的。IWorkbenchPage 除了 findView 方法之外，还有 findEditor 方法是用来得到编辑器的。

像“cn.com.chengang.View2”这种标识符在系统开发中会经常用到，最好建立一个常量专用类来集中放置这些字符串常量，然后系统中用的时候只用其常量名就行了，否则把标识符的字串分散在代码中，以后改起来会让你痛不欲生。

常量类的示意代码如下：

```
public final class StringConstants {
    public final static String VIEW1 = "cn.com.chengang.View1";
    public final static String VIEW2 = "cn.com.chengang.View2";
}
```

要用的时候则这样写：findView(StringConstants.VIEW2);

18.4 给视图加下拉菜单和按钮

本例将给视图加入下拉菜单和按钮，如图 18.6 所示。同时再为列表添加一个右键菜单，这样读者可以用来和视图的下拉菜单进行比较阅读。

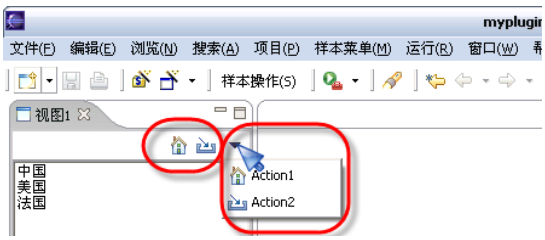


图 18.6 效果图

18.4.1 创建ActionGroup类

加入菜单和按钮的方法与 SWT/JFace 组件的一样。先创建一个 ActionGroup 代码如下:

```
//--------文件名: MyActionGroup.java----------
public class MyActionGroup extends ActionGroup {
    /*
     * 加入按钮
     */
    public void fillActionBars(IActionBars actionBars) {
        if (actionBars == null)
            return;
        IToolBarManager toolBar = actionBars.getToolBarManager();
        toolBar.add(new Action1());
        toolBar.add(new Action2());
    }

    /*
     * 加入下拉菜单、右键弹出菜单
     */
    public void fillContextMenu(IMenuManager menu) {
        if (menu == null)
            return;
        menu.add(new Action1());
        menu.add(new Action2());
    }

    private class Action1 extends Action {
        public Action1() {
           ImageDescriptor
imageDesc=WorkbenchImages.getImageDescriptor(IworkbenchGraphic
Constants.IMG_ETOOL_HOME_NAV);
            setHoverImageDescriptor(imageDesc);
            setText("Action1");
        }
        public void run() {}
    }

    private class Action2 extends Action {
        public Action2() {
            ImageDescriptor              imageDesc              =
WorkbenchImages.getImageDescriptor(Iworkbench
GraphicConstants.IMG_ETOOL_IMPORT_WIZ);
            setHoverImageDescriptor(imageDesc);
            setText("Action2");
        }
        public void run() {}
    }
}
```

程序说明:

- 本程序中含有两个 Action 类: Action1、Action2，和以往的 Action 不同之

处在于它的图像描述符是直接从 Eclipse 环境中取得。既然插件在 Eclipse 环境内运行，那么 Eclipse 环境本身的图标就可以直接拿来使用。

- fillContextMenu 方法比起以前的少了几句。在 18.4.2 小节，可以看到它移出到 View1 类中去了，主要原因是为了此方法兼顾添加视图的下拉菜单。

18.4.2 修改View1 类

在 View1 中增加了三种方法，分别用来加入视图的导航栏按钮、下拉菜单，以及加入列表 List 的右键菜单。代码如下:

```
public class View1 extends ViewPart {
    private List list; //将 List 写成类的实例变量，以扩大它的可访问范围

    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        list = new List(topComp, SWT.BORDER);
        list.add("中国");
        list.add("美国");
        list.add("法国");
        //列表选择事件监听(和以前一样，省略)
        /*
         * 加入导航栏按钮、下拉菜单、右键菜单
         */
        MyActionGroup actionGroup = new MyActionGroup();
        fillViewAction(actionGroup);//加入视图的导航栏按钮
        fillViewMenu(actionGroup);//加入视图的下拉菜单
        fillListMenu(actionGroup);// 加入视图的下拉菜单
    }

    /**
     * 加入视图的导航栏按钮
     */
    private void fillViewAction(MyActionGroup actionGroup) {
        IActionBars bars = getViewSite().getActionBars();
        actionGroup.fillActionBars(bars);
    }

    /**
     * 加入视图的下拉菜单
     */
    private void fillViewMenu(MyActionGroup actionGroup) {
        IMenuManager                    menu                    =
getViewSite().getActionBars().getMenuManager();
        actionGroup.fillContextMenu(menu);
    }

    /**
     * 加入列表 List 的右键菜单
     */
    private void fillListMenu(MyActionGroup actionGroup) {
        MenuManager menu1 = new MenuManager();
```

```
        Menu m = menu1.createContextMenu(list);
        list.setMenu(m);
        actionGroup.fillContextMenu(menu1);
    }

    public void setFocus() {}
}
```

程序说明：

- 过去写在 ActionGroup 中的两句移到了 fillListMenu 方法中。
- 视图加按钮、菜单的方式和以前 SWT/JFace 的方式是一样的。只不过以前用自己生成 MenuManager 对象等，而现在的插件就只需要使用视图已有的 MenuManager 对象。

18.5 加入编辑器（editors）

本节将给出如下的实例：双击视图 1 中的列表项，将在透视图中加入相应的编辑器。这种效果就像在 Eclipse 中双击 Java 源文件，就会打开该源文件相对应的编辑器一样。效果如图 18.7 所示。

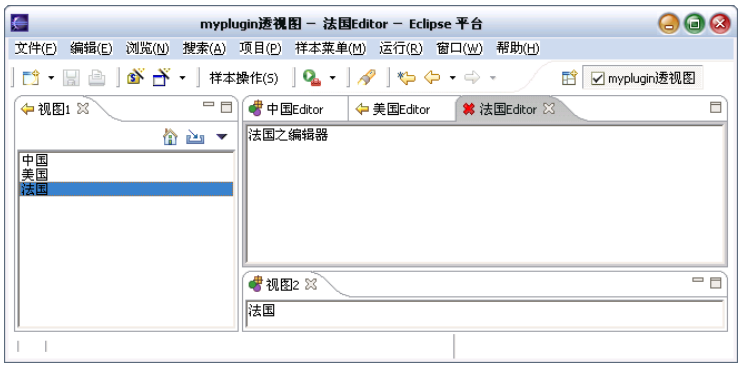


图 18.7 编辑器效果图

和以前一样，先来修改 plugin.xml 文件将编辑器的扩展点加入，然后再创建相应的编辑器类，最后编写列表双击的事件代码。

18.5.1 修改plugin.xml文件，设置三个编辑器的扩展点

```
<extension
    point="org.eclipse.ui.editors">
  <editor
        name="中国 Editor"
        icon="icons/project.gif"
        class="cn.com.chengang.ChinaEditor"
        id="cn.com.chengang.ChinaEditor">
  </editor>
  <editor
        name="美国 Editor"
        icon="icons/prev.gif"
        class="cn.com.chengang.UsaEditor"
        id="cn.com.chengang.UsaEditor">
  </editor>
```

```
  <editor
        name="法国 Editor"
        icon="icons/remove.gif"
        class="cn.com.chengang.FranceEditor"
        id="cn.com.chengang.FranceEditor">
  </editor>
</extension>
```

代码说明:
编辑器的扩展点是 org.eclipse.ui.editors，它各项的含义和视图扩展点基本一样，请参照 18.2.1 小节的视图扩展点的说明。这里强调一点：icon 也是必填项。

18.5.2 创建三个编辑器类

在 18.5.1 小节的 plugin.xml 中，提前设置了编辑器对应的类 cn.com.chengang.ChinaEditor 和 UsaEditor、FranceEditor，本小节就来在包 cn.com.chengang 中创建这三个编辑器类。
编辑器必须实现 IEditorPart 接口,但通常是继承抽象类 EditorPart 类(EditorPart 是 IEditorPart 的子类)。如果继承 EditorPart 则必须实现该抽象类的七种方法，在此先实现方法 init、createPartControl。本例只给出了 ChinaEditor 的代码，UsaEditor、FranceEditor 与之完全类似，ChinaEditor 的代码如下:

```
//---------- 文件名: ChinaEditor.java --------------
public class ChinaEditor extends EditorPart {
    /**
     * Editor 的初始化方法。本方法前两句是固定不变的
     */
    public void init(IEditorSite site, IEditorInput input) throws
PartInitException {
        System.out.println("init");
        setSite(site);
        setInput(input);
        //设置 Editor 标题栏的显示名称。不要,则名称用 plugin.xml 中的 name
属性
        //setPartName(input.getName());
        //设置 Editor 标题栏的图标。不要，则会自动使用一个默认的图标

//setTitleImage(input.getImageDescriptor().createImage());
    }

    /**
     * 在此方法中创建 Editor 中的界面组件
     */
    public void createPartControl(Composite parent) {
        System.out.println("createPartControl");
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        Text text = new Text(topComp, SWT.BORDER);
        text.setText("中国之编辑器");
    }
    //此五个抽象类的方法下面将讲解，现在让它空实现
    public void doSave(IProgressMonitor monitor) {}
    public boolean isSaveAsAllowed() {return false;}
```

```
    public void doSaveAs() {}
    public boolean isDirty() {return false;}
    public void setFocus() {}
}
```

18.5.3 创建IEditorInput

获取视图对象是用 IWorkbenchPage 的 findView 方法,方法参数是视图在 plugin.xml 中的 id 标识。获取编辑器对象是用 findEditor 方法，但该方法的参数却不是 id 标识的字符串，而是一个 IEditorInput 对象。另外，加载一个编辑器是用 IWorkbenchPage 的 openEditor (editorInput, editorID)方法。

由上可知，编辑器都会要对应一个 IEditorInput 和一个 EditorPart，而且在 IWorkbenchPage 中是根据 IEditorInput 来取得 EditorPart，如图 18.8 所示。

图 18.8 编辑器的组成

在本小节将要创建三个 Editor 相对应的 IEditorInput。在这里只给出了 ChinaEditor 对应的 IEditorInput，其他两个与之类似，请读者查阅配书光盘的代码。

```
//--------- 文件名: ChinaEditorInput.java -----------
public class ChinaEditorInput implements IEditorInput {
    /**
     * 返回 true，则打开该编辑器后它会出现在 Eclipse 主菜单"文件"
     * 最下部的最近打开的文档栏中。返回 false 则不出现在其中
     */
    public boolean exists() {
        return true;
    }

    /**
     * 编辑器标题栏的图标，不过它还需要在 ChinaEditor 中用
     * setTitleImage 方法设置，才能出现在标题栏中
     */
    public ImageDescriptor getImageDescriptor() {
        return
WorkbenchImages.getImageDescriptor(IWorkbenchGraphicConstants.
IMG_
ETOOL_HOME_NAV);
    }

    /**
     * 编辑器标题栏的显示名称，和上面的 getImageDescriptor
     * 一样也要在 ChinaEditor 中用 setPartName 方法设置，才能出现在标题栏
中
     */
    public String getName() {
```

```
        return "中国的编辑器";
    }

    /**
     * 编辑器标题栏的小黄条提示文字，不需像 getName 那样在 ChinaEditor 中
再设置
     */
    public String getToolTipText() {
        return "这是视图 1 列表中的中国项对应的编辑器";
    }

    /**
     * 返回一个可以用做保存本编辑输入数据状态的对象，本例让它空实现
     */
    public IPersistableElement getPersistable() {
        return null;
    }

    /**
     * 得到一个编辑器的适配器，本例让它空实现
     *     IAdaptable a = new ChinaEditorInput();
     *     IFoo x = (IFoo)a.getAdapter(IFoo.class);
     *     if (x != null)
     *         [用 x 来做 IFoo 的事情]
     */
    public Object getAdapter(Class adapter) {
        return null;
    }
}
```

18.5.4 打开编辑器

有了 EditorPart 和 IEditorInput 后，就可以在 Eclipse 中打开编辑器了。本例是要实现双击视图 1 的列表，则会打开列表项相对应的编辑器，因此在 View1 类的 List 对象添加一个鼠标双击事件监听器。另外还要考虑到，如果已经打开了列表项对应的编辑器，则下次再双击时就不再打开该项的编辑器，而是将其设成当前选择的编辑器。

查找编辑器的方法是：IEditorPart editor = IWorkbenchPage.findEditor(IEditorInput);

打开编辑器的方法是：IWorkbenchPage.openEditor(IEditorInput, editorID);

所有代码如下：

```
list.addMouseListener(new MouseAdapter() {
    ChinaEditorInput chinaEditorInput = new ChinaEditorInput();
    UsaEditorInput usaEditorInput = new UsaEditorInput();
    FranceEditorInput       franceEditorInput       =       new
FranceEditorInput();

    public void mouseDoubleClick(MouseEvent e) {
        /*
         * 根据列表不同项得到其相应的 editorInput 和 editorID，其中
         * editorID 指该编辑器在 plugin.xml 文件中设置 id 标识值
```

```
         */
        List list = (List) e.getSource();//由 MouseEvent 得到列表对
象
        String listStr = list.getSelection()[0];//得到列表当前项字
符
        IEditorInput editorInput = null;
        String editorID = null;
        if (listStr.equals("中国")) {
            editorInput = chinaEditorInput;
            editorID = "cn.com.chengang.ChinaEditor";
        } else if (listStr.equals("美国")) {
            editorInput = usaEditorInput;
            editorID = "cn.com.chengang.UsaEditor";
        } else if (listStr.equals("法国")) {
            editorInput = franceEditorInput;
            editorID = "cn.com.chengang.FranceEditor";
        }
        //如果 editorInput 或 editorID 为空则中断返回
        if (editorInput == null || editorID == null)
            return;
        //取得 IWorkbenchPage，并搜索使用 editorInput 对象对应的编辑器
        IWorkbenchPage workbenchPage = getViewSite().getPage();
        IEditorPart                    editor                    =
workbenchPage.findEditor(editorInput);
        /*
         * 如果此编辑器已经存在，则将它设为当前的编辑器（最顶端），否则
         * 重新打开一个编辑器
         */
        if (editor != null) {
            workbenchPage.bringToTop(editor);
        } else {
            try {
                workbenchPage.openEditor(editorInput, editorID);
            } catch (PartInitException e2) {
                e2.printStackTrace();
            }
        }
    }
});
```

程序说明:

在本程序中为了便于理解，使用了 if…else 这种简单的方式来判断双击的列表项，这适合列表项较少的情况，如果列表项太多了，则代码就会相当长。解决这个问题，可将各 IEditorInput 对象与列表 List 的项对应起来，并将 eidtorID 写到各 IEditorInput 类中。这样就可以用下面的方式来得到 IEditorInput 和 eidtorID 了。

```
String key = "" + list.getSelectionIndex();
IEditorInput editorInput = (IEditorInput) list.getData(key);
String eidtorID = editorInput.getEditorID();
```

18.5.5 总结

在实际开发中很多界面都是创建在编辑器上，虽然在这里只讲了最常用的编辑器使用方法，但以足够应付大部分开发的需要。如果你想了解更多关于编辑器的信息，可以查阅编辑器的帮助文档，它在帮助中的位置是“平台插件开发者指南→程序员指南→编辑器”。

18.6 编辑器类（EditorPart）方法使用说明

在 18.5 节将 ChinaEditor 类继承 EditorPart 抽象类时，只实现了两种方法：init、createPartControl，本节将通过“EditorPart 方法的执行情况”、“各方法的作用及含义”、“一个实例”来逐步讲解其他的五种方法。

18.6.1 EditorPart方法的执行情况

要使用好 EditorPart，首先得了解其方法在各种情况下的执行流程，我们在类的每一种方法前加上 System.out.println（"方法名：***"），运行后就可以得到如下的结果。
（1）双击列表项打开编辑器时：init→isDirty→createPartControl→isDirty→isDirty→isDirty→isDirty→isDirty→setFocus→isDirty→isSaveAsAllowed。
（2）关闭编辑器时：setFocus→isDirty→isSaveAsAllowed→isDirty→isSaveAsAllowed→setFocus→isDirty，如果保存编辑器，则最后还会执行 doSave 方法。
（3）单击编辑器标题时：setFocus。
（4）编辑器失去焦点时：isDirty→isSaveAsAllowed→isDirty→isSaveAsAllowed。
（5）编辑器得到焦点时：setFocus→isDirty→isSaveAsAllowed→isDirty→isSaveAsAllowed。
（6）当编辑器可以保存，并在主菜单"文件→保存"选项或按 Ctrl+S 键时：isDirty→doSave。

18.6.2 各方法的作用及含义

1. boolean isDirty()

由此方法获知编辑器是否脏了（所谓"脏"是指编辑器中的值已经发生了改变），true 表示脏。当其返回 true 时，会出现两个效果：编辑器的标题前出现一个"*"号，主菜单"文件"下的"保存"项可用。
特别要注意的是，编辑器不会自己判断是否脏了，这需要在程序中用语句手动设置，例如，在编辑器的文本框加入一个键盘监听事件，当在文本框中输入字符时，则将 isDirty 方法返回值设为 true（脏）。
在方法执行情况中，可以看到此方法的执行是最频繁的，所以不要在这种方法中加入太多的执行语句，否则会影响程序执行速度。

2. void doSave ()

在这种方法中编写保存编辑器的代码，当在主菜单中选择"文件→保存"选项时会执行此方法，但在 isDirty 返回 true 时"保存"菜单和 Ctrl+S 键才可以用，也即 isDirty 方法控制着 doSave 方法的执行。
当保存成功时，要注意将脏的状态设回 false，并调用 firePropertyChange 方法将编辑器的界面状态更新（编辑器标题前的"*"号及"保存"菜单）。

3. boolean isSaveAsAllowed()

是否允许编辑器使用"另存为"功能。如果此项返回 false，则不能使用"另存为"功能，而且主菜单"文件"下的"另存为"项被置灰。

4. void doSaveAs()

和 doSave 的作用相似，在这里书写"另存为"功能的处理代码。

5. void setFocus()

当编辑器获得焦点时执行该方法。

18.6.3 一个实例

在这个实例中，当修改 ChinaEditor 编辑器中文本框的文字时，编辑器标题前出现“*”且主菜单“文件”下的“保存”选项可用。当保存了编辑器后，“*”消失并且“保存”菜单不可用。当编辑器为脏时，关闭编辑器会弹出一个提示框（如图 18.9 所示）。

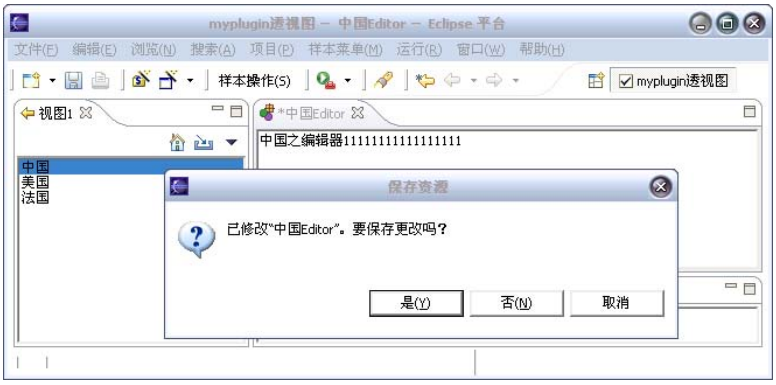


图 18.9 关闭脏编辑器时的效果图

要实现以上效果只需要修改 ChinaEditor 类，修改后的代码如下：

```
public class ChinaEditor extends EditorPart {
    private boolean dirty = true; //编辑器是否为脏的标识

    //……init 方法不变，省略

    /**
     * 在此方法中创建 Editor 中的界面组件
     */
    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        Text text = new Text(topComp, SWT.BORDER);
        text.setText("中国之编辑器");
        text.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                //如果编辑器不脏（即没有修改），则标志它脏并刷新界面状态
                if (!isDirty()) {
                    setDirty(true);
                    firePropertyChange(IEditorPart.PROP_DIRTY);
                }
            }
        });
    }

    /**
     * 保存的处理代码在这种方法中，当按 Ctrl+S 键时会执行此方法。
     * 最后别忘记标志为非脏及刷新界面状态
     */
    public void doSave(IProgressMonitor monitor) {
        if (isDirty()) {
```

```
            //……保存编辑器事件处理代码（省略）
            setDirty(false);
            firePropertyChange(IEditorPart.PROP_DIRTY);
        }
    }

    /**
     * 是否允许“另存为”
     */
    public boolean isSaveAsAllowed() {
        return false; //不允许
    }

    /**
     * “另存为”的代码写在这里，本例不实现它
     */
    public void doSaveAs() {}

    /**
     * dirty 标识的 set 方法，由此方法设置编辑器为脏
     */
    public void setDirty(boolean dirty) {
        this.dirty = dirty;
    }

    /**
     * 编辑器的内容是否脏了。true 脏,false 不脏
     */
    public boolean isDirty() {
        return dirty;
    }

    /**
     * 当编辑器获得焦点时会执行此方法，本例空实现
     */
    public void setFocus() {}
}
```

程序说明:

firePropertyChange(IEditorPart.PROP_DIRTY);这一句除了能将界面状态刷新之外,如果 IEditorPart 添加了如下的监听器,则还可以触发其中的 propertyChanged 事件。

```
chinaEditor.addPropertyListener(new IPropertyListener() {
    //此时 source 为 ChinaEditor 对象,propId 为 IEditorPart.PROP_DIRTY 这个常量值
    public void propertyChanged(Object source, int propId) {
        //事件处理代码，省略
    }
});
```

18.7 加入首选项（preferencePages）

选择主菜单“窗口→首选项”选项打开“首选项”窗口，如图 18.10 所示。这个窗口是 Eclipse 所有设置项的集中地，同样也常是第三方插件进行设置的窗口。图中左边两个方框标注的就是 SWT Designer 插件的设置树和本书插件 MyPlugin 的设置树。
本节将实现 MyPlugin 的首选项中的设置树。

图 18.10 “首选项”窗口

18.7.1 修改plugin.xml文件，设置首选项的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>项之前：

```
<extension point="org.eclipse.ui.preferencePages">
    <page
      name="myplugin2 插件设置"
      class="cn.com.chengang.preferences.RootPreferencePage"
      id="cn.com.chengang.preferences.RootPreferencePage">
    </page>
    <page
      name="DB 数据库"
      category="cn.com.chengang.preferences.RootPreferencePage"
      class="cn.com.chengang.preferences.DBPreferencePage"
      id="cn.com.chengang.preferences.DBPreferencePage">
    </page>
</extension>
```

代码说明：

- org.eclipse.ui.preferencePages 是首选项的扩展点。
- name 是首选项的树结点的名称。
- class 是首选项的树结点所对应的类（还没编写，下一步将完成此类）。
- id 是首选项的树结点标识，建议设置成和 class 一样的名称。
- category 是要等于父结点的 id 标识。

18.7.2 建立首选项各结点对应的类

在 18.7.1 小节的 plugin.xml 中提前设置了首选项结点对应的类 RootPreferencePage、DBPreferencePage，本小节就来在包 cn.com.chengang.preferences 中创建此类。
首选项的类必须继承 PreferencePage 抽象类和实现 IWorkbenchPreferencePage 接

口，实现接口只有一种方法 init，抽象类则有一些“首选项”窗口的按钮的执行方法。本小节实例先给出代码简单一些的根结点 RootPreferencePage 类，再给出复杂一些的子结点 DBPreferencePage 类，两类具体代码如下：

```
//---------文件名: RootPreferencePage.java---------------------
public class RootPreferencePage extends PreferencePage
                                    implements
IWorkbenchPreferencePage {

   public void init(IWorkbench workbench) {}

   protected Control createContents(Composite parent) {
      Composite topComp = new Composite(parent, SWT.NONE);
      topComp.setLayout(new RowLayout());
      new Label(topComp, SWT.NONE).setText("欢迎使用 myplugin2 插
件");
      return topComp;
   }
}
```

```
//---------文件名: DBPreferencePage.java---------------------
public class DBPreferencePage extends PreferencePage
                  implements         IWorkbenchPreferencePage,
ModifyListener {

   //为文本框定义三个键值
   public static final String URL_KEY = "$URL_KEY";
   public static final String USERNAME_KEY = "$USERNAME_KEY";
   public static final String PASSWORD_KEY = "$PASSWORD_KEY";
   //为文本框值定义三个默认值
   public     static     final     String     URL_DEFAULT     =
"jdbc:db2://127.0.0.1/mydb";
   public static final String USERNAME_DEFAULT = "glchengang";
   public static final String PASSWORD_DEFAULT = "12345678";
   //定义三个文本框
   private Text urlText;
   private Text usernameText;
   private Text passwordText;
   //定义一个 IPreferenceStore 对象
   private IPreferenceStore ps;

   /**
    * 接口 IWorkbenchPreferencePage 的方法，负责初始化。在此方法中设置
一个
    * PreferenceStore 对象，由此对象提供文本框值的读入/写出方法
    */
   public void init(IWorkbench workbench) {


setPreferenceStore(Myplugin2Plugin.getDefault().getPreferenceS
tore());
   }
```

```
   /**
    * 父类的界面创建方法
    */
   protected Control createContents(Composite parent) {
      Composite topComp = new Composite(parent, SWT.NONE);
      topComp.setLayout(new GridLayout(2, false));
      /*
       * 创建三个文本框及其标签
       */
      new Label(topComp, SWT.NONE).setText("URL:");
      urlText = new Text(topComp, SWT.BORDER);
      urlText.setLayoutData(new
GridData(GridData.FILL_HORIZONTAL));

      new Label(topComp, SWT.NONE).setText("用户名:");
      usernameText = new Text(topComp, SWT.BORDER);
      usernameText.setLayoutData(new
GridData(GridData.FILL_HORIZONTAL));

      new Label(topComp, SWT.NONE).setText("密码:");
      passwordText   =   new   Text(topComp,   SWT.BORDER   |
SWT.PASSWORD);
      passwordText.setLayoutData(new
GridData(GridData.FILL_HORIZONTAL));
      //取得一个 IPreferenceStore 对象
      ps = getPreferenceStore();
      /*
       * 取出以前保存的值，并将其设置到文本框中，如果取出值为空
       * 或者是空字串，则填入默认值
       */
      String url = ps.getString(URL_KEY);
      if (url == null || url.trim().equals(""))
         urlText.setText(URL_DEFAULT);
      else
         urlText.setText(url);

      String username = ps.getString(USERNAME_KEY);
      if (username == null || username.trim().equals(""))
         usernameText.setText(USERNAME_DEFAULT);
      else
         usernameText.setText(username);

      String password = ps.getString(PASSWORD_KEY);
      if (password == null || password.trim().equals(""))
         passwordText.setText(PASSWORD_DEFAULT);
      else
         passwordText.setText(password);
      /*
       * 添加事件监听，this 代表本类，因本类也实现了 ModifyListener 接
口，
       * 所以本类可以作为监听器使用
```

```
         */
        usernameText.addModifyListener(this);
        passwordText.addModifyListener(this);
        urlText.addModifyListener(this);
        return topComp;
    }

    /**
     * 本类实现 ModifyListener 接口的方法，当三个文本框中发生修改时将执行此方法。
     * 方法中对输入值进行了验证并将“确定”、“应用”两按钮使能
     */
    public void modifyText(ModifyEvent e) {
        String errorStr = null;//将原错误信息清空
        if (urlText.getText().trim().length() == 0) {
            errorStr = "URL 不能为空！";
        } else if (usernameText.getText().trim().length() == 0) {
            errorStr = "用户名不能为空！";
        } else if (passwordText.getText().trim().length() == 0) {
            errorStr = "密码不能为空！";
        }
        setErrorMessage(errorStr);//errorStr=null 时复原为正常的提示文字
        setValid(errorStr == null);//“确定”按钮
        getApplyButton().setEnabled(errorStr == null);//“应用”按钮
    }

    /**
     * 父类方法，单击“复原默认值”按钮时将执行此方法，取出默认值设置到文本框中
     */
    protected void performDefaults() {
        urlText.setText(URL_DEFAULT);
        usernameText.setText(USERNAME_DEFAULT);
        passwordText.setText(PASSWORD_DEFAULT);
    }

    /**
     * 父类方法，单击“应用”按钮时执行此方法，将文本框值保存并弹出成功的提示信息
     */
    protected void performApply() {
        doSave(); //自定义方法，保存设置
        MessageDialog.openInformation(getShell(), "信息", "成功保存修改!");
    }

    /**
     * 父类方法，单击“确定”按钮时执行此方法，将文本框值保存并弹出成功的提示信息
     * @return true 成功退出
     */
```

```
    public boolean performOk() {
        doSave();
       MessageDialog.openInformation(getShell(),"信息","修改在下次
启动生效");
        return true;
    }

    /**
     * 自定义方法，保存文本框的值
     */
    private void doSave() {
        ps.setValue(URL_KEY, urlText.getText());
        ps.setValue(USERNAME_KEY, usernameText.getText());
        ps.setValue(PASSWORD_KEY, passwordText.getText());
    }
}
```

18.7.3 运行插件

运行插件后，打开新 Eclipse 环境的“首选项”窗口，选择“DB 数据库”选项，将其中的密码删除后可得到如图 18.11 所示的出错效果。

图 18.11　首选项出错的效果图

18.7.4　总结

本例程的核心是 IPreferenceStore 对象的使用，用它的 getString 方法来取值、setValue 方法来存值。其次和以前的事件代码写法有所不同的是：本类实现了 ModifyListener 接口，也成为了一个监听器，这样在各文本框的加入监听器的代码就会简洁很多，不过其事件代码必须保证三个文本框可以共用才行。

18.8　加入帮助（toc）

如图 18.12 所示，本节将为 myplugin2 插件加入帮助。本节实例将演示如何在 plugin.xml 添加扩展点，如何创建帮助左部的结点树，如何链接到帮助文件。

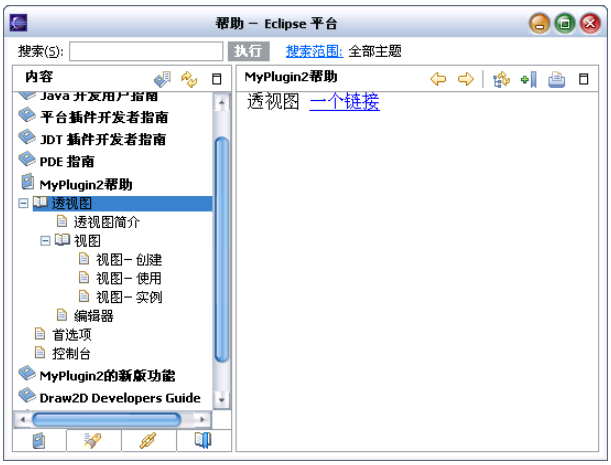


图 18.12　帮助

18.8.1　修改plugin.xml文件，设置三个帮助的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>项之前：

```
<extension point="org.eclipse.help.toc">
  <toc
        file="toc.xml"
        primary="true">
  </toc>
  <toc
        file="other_toc.xml"
        primary="true">
  </toc>
</extension>
```

说明：

- org.eclipse.help.toc 是帮助的扩展点。
- toc 是帮助目录项的设定。
- toc 的 file 是指定帮助目录文件（还没编写，18.8.2 小节将完成这两个 xml 文件）。
- toc 的 primary 是该项帮助目录是否显示在帮助窗口中，true 为显示。
- 如图 18.12 所示左边的结点树，toc.xml 对应于“myplugin2 帮助”结点，other_toc.xml 对应于“myplugin2 的新版功能”结点。

18.8.2 编写帮助目录文件toc

这两个文件应放置在项目的根目录下，和 src 目录平级，如图 18.13 所示。

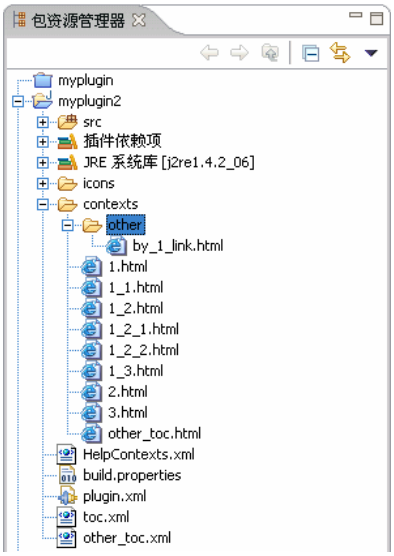


图 18.13 目录结构图

先给出第一个文件 toc.xml 的代码，如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<toc label="myplugin2 帮助">
    <topic label="透视图" href="contexts/1.html">
        <topic label="透视图简介" href="contexts/1_1.html"/>
        <topic label="视图" href="contexts/1_2.html">
            <topic label="视图－创建" href="contexts/1_2_1.html"/>
            <topic label="视图－使用" href="contexts/1_2_2.html"/>
            <topic label="视图－实例" href="contexts/1_2_3.html"/>
        </topic>
        <topic label="编辑器" href="contexts/1_3.html"/>
    </topic>
    <topic label="首选项" href="contexts/2.html"/>
    <topic label="控制台" href="contexts/3.html"/>
    <topic label="插件网站" href="http://blog.csdn.net/glchengang"/>
</toc>
```

说明:

- topic: 设置帮助目录中的结点。
- topic 的 lable: 结点显示的名称。
- topic 的 href: 结点对应的帮助文件的路径及文件名，也可以是网址（还没编写对应的帮助文档，18.8.1 小节将完成它们）。
- 这里应注意一个细微的地方: xml 中项的结束符有两种，如
  <topic label="首选项" href="contexts/2.html"/>
  或者是:
  <topic label="首选项" href="contexts/2.html">< /topic>
  这两者的效果是一样的。

下面再给出另一个帮助目录文件 other_toc.xml，代码如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<toc label="myplugin2 的新版功能">
   <topic label="自制绘图" href="contexts/other_toc.html"/>
</toc>
```

18.8.3 创建相应的帮助文档

帮助文件创建在插件项目的根目录下，和 src 目录平级，目录结构如图 18.13 所示。帮助文件都是一些标准的 html 格式的网页文件，这里仅给出<topic label="透视图" href="contexts/1.html">项对应的 1.html 文件，其内容如下:

```
<HTML>
   <BODY>
      透视图 <A HREF="other/by_1_link.html">一个链接</A>
   </BODY>
</HTML>
```

在 1.html 中，有一个链接“other/by_1_link.html”，这个文件并不需要在帮助目录文件 toc.xml 中注册。

18.8.4 总结

创建帮助一般都分三步走:
（1）在 plugin.xml 添加扩展点。
（2）编写帮助目录的 toc 文件。
（3）编写相应的帮助文档。
还有一种做法是将帮助单独写成一个插件，这种插件和 myplugin2 插件没有什么不同，其开发过程也和本节一样，只不过该插件里只含有帮助的文件。这种方式适合比较大的项目使用，因为分成了没有什么联系的两个插件，这样就可以让一批人开发 myplugin2 插件，另一批人开发帮助插件，两队人马互不干扰。

18.9 弹出信息式的帮助（contexts）

弹出信息可以和窗口组件关联在一起，给用户显示有针对性的帮助，其效果如图 18.14 所示。如果组件设置有弹出信息，按 F1 键就可以激活它。

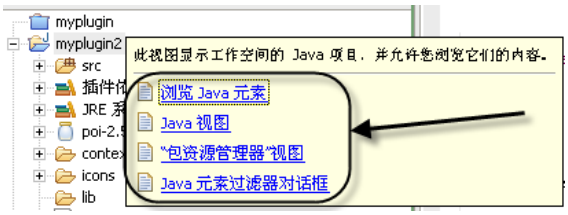


图 18.14　弹出信息

下面就以一个实例来演示如何实现弹出信息。

18.9.1　修改plugin.xml文件，设置弹出信息的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>项之前。

```
<extension point="org.eclipse.help.contexts">
   <contexts
         file="HelpContexts.xml">
   </contexts>
</extension>
```

代码说明：

- org.eclipse.help.contexts 是弹出信息的扩展点。
- contexts 的 file 指定弹出信息的设置文件（尚未编写，下一步将完成此 xml 文件）。

18.9.2 编写弹出信息的设置文件HelpContexts.xml

HelpContexts.xml 文件应放置在项目的根目录下，和 src 目录平级。

```
<?xml version="1.0" encoding="UTF-8"?>
<contexts>
   <context id="textHelpId">
      <description>编辑器</description>
      <topic href="contexts/1_3.html"  label="编辑器的帮助"/>
   </context>
   <context id="buttonHelpId">
      <description>视图</description>
      <topic href="contexts/1_2_1.html"  label="视图－创建"/>
      <topic href="contexts/1_2_2.html"  label="视图－使用"/>
      <topic href="contexts/1_2_3.html"  label="视图－实例"/>
   </context>
</contexts>
```

代码说明:

- 这里设置了两个弹出信息 textHelpId、buttonHelpId。
- id: 弹出信息的标识。
- description: 弹出信息时的标题栏文字。
- href: 弹出信息子项所对应的帮助文件，可以和 18.8 节的帮助文件共用，也可以自己设定单独的帮助文件。
- label: 弹出信息子项的显示名称。

HelpContexts.xml设置的效果如图 18.15 所示。

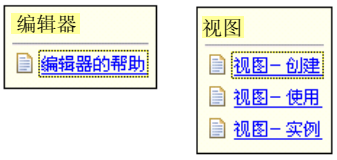


图 18.15 HelpContexts.xml 设置的效果图

18.9.3 创建弹出信息对应的帮助文件

由于在 HelpContexts.xml 中，href 项设置的都是 18.8 节的帮助文件，所以本例此步省略。

18.9.4 在界面组件中设置弹出信息

在界面组件中设置弹出信息分成以下三步。

（1）弹出信息的准备工作已经完成，接下来是将它们和对应组件关联起来。方法如下:

```
WorkbenchHelp.setHelp(text, "myplugin2.textHelpId");
```

第一个参数是组件对象名，第二个参数是弹出信息的 id 标识，前面要加上 plugin.xml 的<plugin>项设置的 id 标识。

（2）创建一个自定义 Dialog 来演示弹出信息。代码如下:

```
//---------- 文件名: HelpContextsDialog.java -----------
public class HelpContextsDialog extends Dialog {
   protected HelpContextsDialog(Shell parentShell) {
      super(parentShell);
   }
```

```
    protected Control createDialogArea(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new RowLayout());
        //界面组件
        Text text = new Text(topComp, SWT.BORDER);
        text.setText("编辑器           ");
        Button button = new Button(topComp, SWT.BORDER);
        button.setText("  打开视图  ");
        //让弹出信息和界面组件关联起来
        WorkbenchHelp.setHelp(text, "myplugin2.textHelpId");
        WorkbenchHelp.setHelp(button, "myplugin2.buttonHelpId");
        return topComp;
    }
}
```

（3）编写打开此 Dialog 的方法，本例选择视图 1 上的一个按钮来打开 Dialog。在 MyActionGroup 类的 Action1 中的 run 方法中加入如下语句：

```
HelpContextsDialog dialog = new HelpContextsDialog(null);
dialog.open();
```

18.9.5 运行插件

运行插件后的效果如图 18.16 所示。

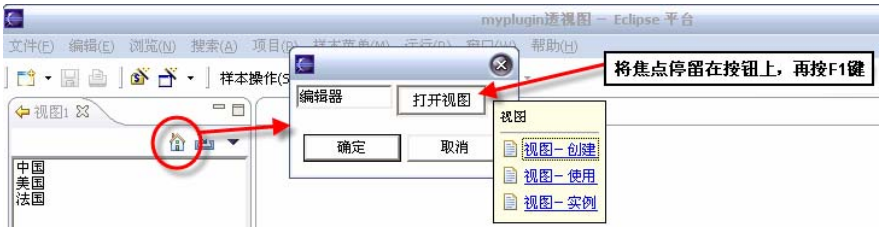


图 18.16 实例的弹出信息界面

18.9.6 总结

设置弹出信息和设置帮助（toc）有其类似之处，而且它们的 HTML 格式的帮助文件也可以共用，但这并不是说弹出信息要依赖于帮助（toc）的设置。而且不是所有的组件都支持弹出信息。下面给出不支持弹出信息的组件例表：

- ToolItem;
- CtabItem;
- TabItem;
- TableColumn;
- TableItem;
- TableTreeItem;
- TreeItem。

18.10 本 章 小 结

本章介绍了插件中最常用的扩展点，不仅讲述了扩展点的含义，而且给出相应的实例和具体

的实现步骤。本章的每节组合在一起就是一个插件开发的开发流程，从透视图开始到帮助结束，插件开发大致是按照这个流程来走。

第 21 章　项目打包与发行

当项目完成后接下来的就是打包发行了，应用程序（Application）项目和 Eclipse 插件项目（plugin）的打包是不同的，本章将分别介绍两者的打包方法，并给出实际的打包例子。

7.1　应用程序项目的打包与发行

7.1.1　简介

Java 应用程序项目完成后是可以脱离 Eclipse 运行的，要运行程序先要打它打成一个 JAR 包，它打包的大部份方法和标准 Java 的 AWT/SWING 的打包方法一样，主要有以下几个要点

MANIFEST.MF － 打包清单。它是打包的关键性文件，主要是设置执行入口类和支持库的路径，在运行 Java 应用程序时是要根据此文件中给出的信息来查找入口类和支持库。

支持包 －如果 Java 应用程序用到了一些 Eclipse 包，那么就必须将这些包也复制到程序运行目录，否则程序将无法运行。如 swt 组件支持包 swt.jar，jface 组件支持包 jface.jar。这些包都要在 MANIFEST.MF 文件中设置好。

本地化文件 － 如果用到了 SWT 组件，则还需要将 SWT 的本地化文件 swt-win32-3063.dll（3063 是版本号）复制到程序运行目录，否则程序将无法运行。

7.1.2　打包的具体操作步骤

本节将用前几章开发的 SWT/JFace 项目“myswt”的打包为例，来介绍打包应用程序项目的方法。

1、编辑清单 MANIFEST.MF

（1）Eclipse 提供了用于打包项目的“导出”向导，但本例运行此向导之前先需要创建一个 MANIFEST.MF 清单文件，其内容如下:

```
Manifest-Version: 1.0
Main-Class: book.chapter_4.wizard_dialog.WizardDialog1
```

Class-Path: ./lib/swt.jar ./lib/jface.jar ./lib/runtime.jar

说明:

Manifest-Version - 指定清单文件的版本号

Main-Class - 指定程序运行的入口类。本例设为运行 4.5.2 节开发的向导式对话框。注意: 类名后不要加 class 扩展名

Class-Path - 指定支持库的路径。“.”指程序运行目录，即导出的 JAR 包所在目录。程序运行时依据 Class-Path 项的设置路径来查找支持库。每一个支持库之间用空格隔开。在这里 jface.jar 需要用到 runtime.jar 包，所以 runtime.jar 包也要加入到 Class-Path 中。

除了入口类的包名和类名之外，其他设置项都不分大小写，比如: Class-Path 写成 class-path 或 CLASS-PATH 也可以，swt.jar 写成 SWT.JAR 也行。

（2）将清单文件保存下来，建议放在 myswt 项目的根目录下。它的文件名可以任意取，本例取名为 manifes.txt，Eclipse 向导在打包时会自动的将 manifes.txt 的内容复制到 JAR 包的 META-INF 目录下的 MANIFEST.MF 文件中。

2、使用 Eclipse“导出”向导来打包项目

（1）右键单击 myswt 项目的项目名，在弹出菜单中选择“导出”。在弹出的如下图 7.1 所示的对话框中，选择“JAR 文件”，单击“下一步”。

图 7.1 导出对话框

（2）如下图 7.2 所示，将右边不需要的文件都取消勾选。在“选择导出目标”项文本框中设置 JAR 包的输出路径和包名（可以任意取名）为“D:\myswt_application\myswt.jar”。接受其他的默认设置不变，单击“下一步”。

附注: 左边虽然选择了 src 目录，但源文件并不会导出到包中，除非勾选了“导出 Java 源代码文件和资源”项。

图 7.2　选择导入文件

（3）如下图 7.3 所示，接受默认设置不变，单击“下一步”。

图 7.3　导出类的选项

（4）这一步较关键。如下图 7.4 所示，选择“从工作空间中使用现有清单”项，将创建的清单文件输入，也可以通过旁边的“浏览”按钮来选择清单文件。输入清单文件后，单击“完成”，Eclipse 开始将项目打包。

图 7.4　清单文件设置

经过以上四步后，在“D:\myswt_application”路径下生成了一个名为“myswt.jar”的文件。myswt.jar 是一个 ZIP 格式的压缩文件，可以用 WinRAR 或 WinZip 软件打开，也就是说用这两个软件也可以替代 Eclipse 向导来打包文件。如果用 WinRAR 来打包文件，则压缩格式要选择 ZIP 格式而非 RAR 格式，压缩率倒可以任意选。

用 WinRAR 打开 myswt.jar 文件后其内部的目录结构如下图 7.5 所示：

图 7.5　myswt.jar 文件的内部目录结构

在 myswt.jar 文件的内部目录 META-INF 中仅一个文件：MANIFEST.MF，它和以前创建的清单文件 manifest.txt 的内容是一样的，如下：

```
Manifest-Version: 1.0
```

```
Class-Path: ./lib/swt.jar ./lib/jface.jar ./lib/runtime.jar

Main-Class: book.chapter_4.wizard_dialog.WizardDialog1
```

3、复制 Java 应用程序的支持包及本地化文件

在 MANIFEST.MF 文件中的 Class-Path 项设置了三个包，从 Eclipse 的 plugins 目录中将此三个支持包复制到 D:\myswt_application\lib 目录，本地化文件 swt-win32-3063.dll 复制到 D:\myswt_application 目录中。此三个文件在 Eclipse 中的路径为:

```
plugins\org.eclipse.swt.win32_3.0.1\ws\win32\swt.jar

plugins\org.eclipse.jface_3.0.0\jface.jar

plugins\org.eclipse.core.runtime_3.0.1\runtime.jar

plugins\org.eclipse.swt.win32_3.0.1\os\win32\x86\swt-win32-3063.dll
```

复制完成后的目录结构如下图 7.6 所示:

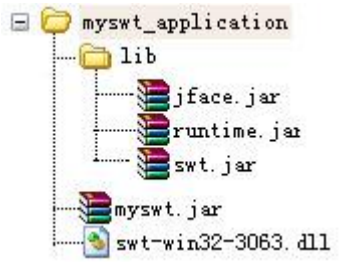


图 7.6　目录结构图

4、编写运行 myswt.jar 包的批处理程序“run.bat”

在 myswt_application 目录下创建一个批处理程序 run.bat（名字任取，扩展名必须是 bat），其内容仅一句语句，如下:

```
javaw -jar myswt.jar
```

说明:

javaw 对应 c:\jdk\jre\bin\javaw.exe 文件，如果 windows 提示命令未发现，则需要将 c:\jdk\jre\bin 路径加入到 windows 环境变量 path 中。

在运行程序的时候有一个讨厌的黑色命令行窗口，要去掉它，可以将 run.bat 内容更改如下:“start javaw -jar myswt.jar”，start 是指调用了 windows 的“运行”命令。

如果想将 swt-win32-3063.dll 也放在单独的目录中，如“D:\myswt_application\native”目录，则需将 run.bat 内容更改为:

```
start javaw -Djava.library.path=./native/ -jar myswt.jar
```

5、运行程序

双击 run.bat 文件，得到如下图 7.7 所示的程序界面。

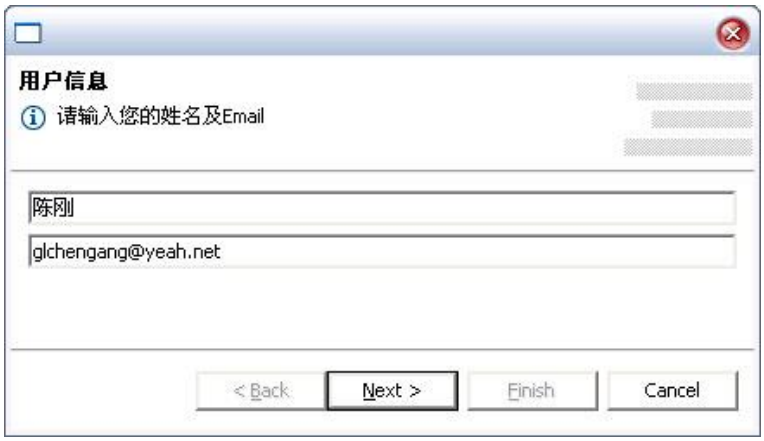


图 7.7　程序运行效果图

6、注意事项

本例只需要三个支持包，但你的程序也许会需要更多的支持包才能运行。如果你想一次到位，则可以将“Java 构建路径”的“库”选项卡中所有引用的包都复制到 lib 目录中。如果你喜欢用到什么包才加入什么包，希望维持打包文件的简洁，则需要自己一步步的去试：如果缺少某支持包，运行程序时会输出的未找到类的错误信息，从信息中的包名可得知程序缺少哪一个支持包。比如“Exception in thread "main" java.lang.NoClassDefFoundError: org/eclipse/jface/wizard/IWizard”，从错误信息中很明显的就能知道程序缺少 jface 包

7.1.3　其他得到 JAR 包的方式

要得到 JAR 包除了以上所说的用 Eclipse“导出”向导、用 WinZip 和 WinRAR，另外还能用 Java 自带的命令行式打包软件 jar.exe（位于 c:\jdk\bin 目录），其打包命令为：

```
c:\jdk\bin\jar cvfm myswt.jar
C:\eclipse3.0.1\eclipse\workspace\myswt\manifest.txt -C
C:\eclipse3.0.1\eclipse\workspace\myswt\bin .
```

说明：

c:\jdk\bin\jar – 由于本书没有把 c:\jdk\bin 加入到 windows 环境变量 path 中，所以手工指定 jar.exe 的路径

cvfm – jar.exe 的参数，“c”创建新的 jar 包；“v”将调试信息打印在屏幕上；“f”指定生成的 jar 文件名；“m”使用清单文件。注意它们都是小写

myswt.jar – 打包后的 JAR 包名

在前面是把清单文件 manifest.txt 放在 C:\eclipse3.0.1\eclipse\workspace\myswt\目录下。如果将它和批处理文件放在一个目录就不必指定长长的路径了。

“-C 路径 .”指将路径下（包括子目录）的所有文件打包，由于 class 文件是输出在项目的 bin 目录下，所以路径指定到项目的 bin 目录，注意三者之间是用空格隔开，并且最后一个字符是小数点。

这种方式的优点是没有 Eclipse 导出向导的操作那么麻烦，适合经常需要导出 JAR 包的情况。

7.1.4 使用第三方插件对项目打包

开源组织（http://sourceforge.net/）中有一款可将 Eclipse 支持包和项目编译文件一起打到一个包中的插件，叫“Fat Jar”，它的下载地址是“http://fjep.sourceforge.net/”，具体下载不再说明，安装步骤参阅第 1 章 SWT Designer 的安装。

Fat Jar 的使用步骤如下:

（1）右键单击 myswt 项目的项目名，可见菜单中多了一项“Build Fat Jar”，如下图 7.8 所示，选择“Build Fat Jar”项。

图 7.8 右键菜单

（2）在下图 7.9 所示的对话框中，“Jar-Name”项填入 JAR 包的输出路径。文件清单“Manifest”项不用填，默认会自动创建一个。“Main-Class”项填入程序的入口类。其他都接受默认值，单击“下一步”。

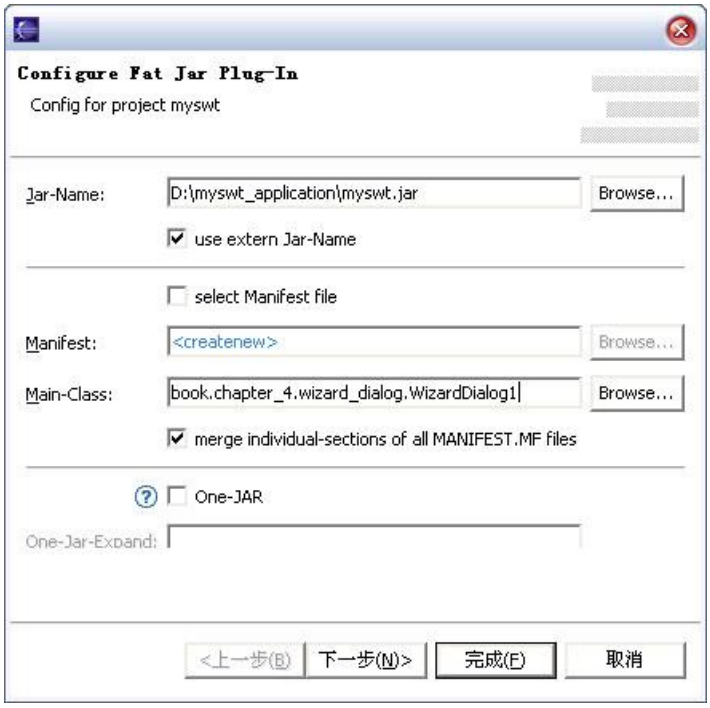


图 7.9　输出配置

（3）如下图 7.10 所示，窗口中将 myswt 项目所用到的支持包都列了出来。我们仅勾选图中 runtime.jar、swt.jar、jface.jar 这三项即可，当然全选也并尝不可，只是最后得到的 JAR 包会更大一些，因为 Fat Jar 会将所有支持包合并在一个 JAR 包中。

图 7.10　选择要打包的文件

单击图 7.10 的“完成”按钮后， JAR 包 myswt.jar 将输出到 D:\myswt_applicationh 目录中。和以前一样，要运行此 JAR 包需要一个批处理文件以及本地化文件 swt-win32-3063.dll，唯一不同的是不再需要 Eclipse 支持包，其目录结构如下图 7.11 所示:

图 7.11　目录结构

为什么不需要 Eclipse 支持包了呢？那是因为支持包已经在 myswt.jar 文件中了，从下图 7.12 可以看到 swt.jar 等都被拆散成目录，并包含在 myswt.jar 包中。

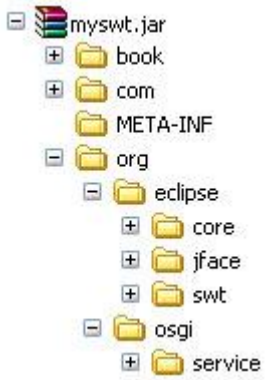


图 7.12　myswt.jar 的内部目录结构

其中 META-INF 目录的 MANIFEST.MF 文件内容如下，可以发现和以前不同的地方是：Class-Path 项没有了。

```
Manifest-Version: 1.0

Created-By: Fat Jar Eclipse Plug-In

Main-Class: book.chapter_4.wizard_dialog.WizardDialog1
```

7.1.4　让用户电脑不必安装 JRE 环境

通常运行 Java 程序有个前提条件：用户电脑必须先安装 JRE 环境。虽然安装 JRE 环境非常简单，但毕竟多了一步，算是有一点点的瑕疵。这里给出一个不必让用户安装 JRE 环境的方法，其实现步骤如下：

（1）将原 JDK 中的“jre”目录复制到“D:\myswt_application\java1.4.2”目录下（java1.4.2 也可换成其他名称）。

（2）将 JDK 和 JRE 从本机卸载掉，这样表示本机没有安装 JAVA 运行环境。

（3）修改批处理文件 run.bat 中的命令为“start java1.4.2\jre\bin\javaw -jar myswt.jar”，仅仅是在 javaw 前加上了一个相对应路径。

双击 run.bat 即可在不安装 JRE 环境的电脑运行此 Java 应用程序。

7.1.5　更进一步的完善

1、抛弃批处理文件（*.bat）

用批处理文件运行程序似乎不够专业，虽然它足以完成运行任务。但习惯就象一种毒药一旦染上就很难摆脱它的影响，Windows 统治下的人们早已经习惯运行扩展名是 EXE 的程序，用*.bat 他们就会感觉别扭。

我们可以用一个叫 JavaLauncher 的免费小程序来代替批处理文件去运行 Java 程序。JavaLauncher 的下载网址是:

http://www.rolemaker.dk/nonRoleMaker/javalauncher/marner_java_launcher.htm

下载下来的文件是一个名 JavaLauncher.zip 的压缩包，解压后的目录结构如下图 7.13 所示:

图 7.13　JavaLauncher.zip 目录结构

在上图的目录中

source 目录包含了 JavaLauncher 的源程序，是用 C 语言写的

changes.txt 是新版的修改说明

launch.exe 是主程序

launcher.cfg 是配置文件

readme.txt 是一些说明和示例

我们只需要 launch.exe、launcher.cfg 两个文件，将这两个文件复制到打包文件所在的目录。launcher.cfg 是一个仅三行内容的文本文件，将它修改如下:

```
.
.\java1.4.2\jre\bin\javaw.exe
-jar myswt.jar
```

第一行设置指向 JAR 包 myswt.jar 的目录，由于 launch.exe 和 myswt.jar 同在一个目录，所以用“.”即当前目录。

第二行设置指向 jre\bin\javaw.exe 的路径。在上一小节（7.1.4 节）已将 jre 目录复制到了 java1.4.2 子目录中

配置好 launcher.cfg 后，双击 launch.exe 即可运行 java 应用程序。

如果仔细研究 eclipse 的启动方式，发现 eclipse 和 JavaLauncher 的原理一样：eclipse.exe 相当于 launch.exe，startup.jar 相当于 myswt.jar。只不过 eclipse.exe 不象 launch.exe 要具有通用性，所以它没有*.cfg 这样的配置文件，而是将启动信息固化在 eclipse.exe 中。

2、美化图标

launch.exe 文件的图标太单调了，让我们给它换个好看点的。换程序的图标需要用到一个免费的软件：Resource Hacker，它有中文版，下载网址是：

http://www.users.on.net/johnson/resourcehacker/

用 Resource Hacker 来替换 launch.exe 的图标的步骤如下：

（1）运行 Resource Hacker，得到如下图 7.14 所示的窗口。

图 7.14 Resource Hacker 的主界面

（2）单击主菜单“文件→打开”，将 launch.exe 载入到程序中，结果如下图 7.15 所示。

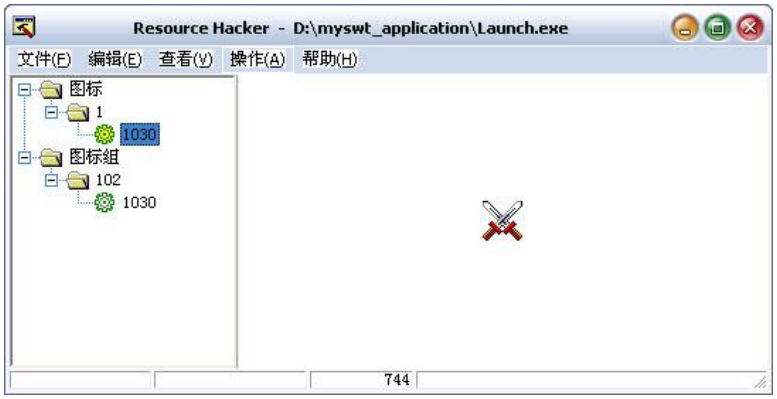


图 7.15 载入 Lanunch.exe 之后的界面

（3）如上图，选择左边的“图标→1→1030”，然后右键单击“1030”项，选择“替换资源...”。如下图 7.16 所示，在弹出窗口中单击“打开新图标文件”，选择一个满意的图标，然后单击“替换”按钮。

附注：图标文件可以是 exe、dll、res、ico，该软件可以从 exe、dll、res 抽取出图标，本例选择的是 java 的一个图标文件 JavaCup.ico。

图 7.16　选择图标文件

（4）如下图 7.17 所示，选择“文件→另存为”，取名 myswt.exe。

附注：按理说选择“保存”也是可以的，这时 Resource Hacker 会将老的 launch.exe 备份成 launch_original.exe。但也许是刷新上有问题，用“保存”方式有时 launch.exe 无法显示出新图标，但有时又可以。

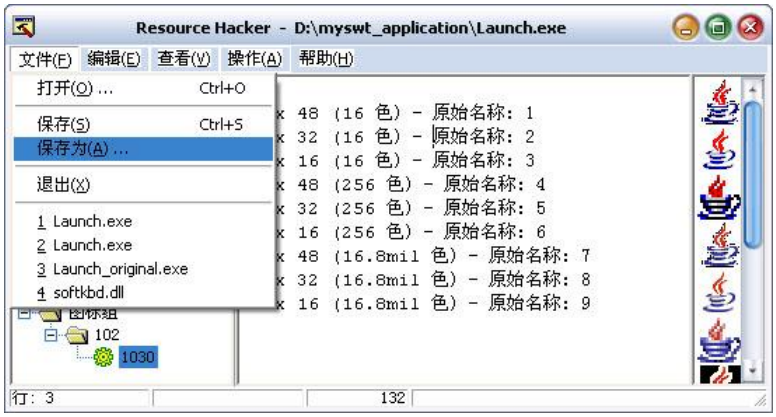


图 7.17　保存修改

（5）最后的目录如下图 7.18 所示，可见 myswt.exe（也就是 launch.exe 改了个名称）的图标换成了 Java 的图标。双击 myswt.exe 即可运行 Java 应用程序。

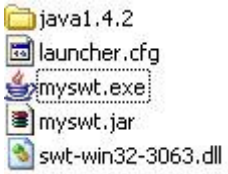


图 7.18　最后的效果

3、最后的打包

发送给用户之前，通常要用 WinZip 或 WinRAR 将所有的文件全部打成一个压缩包，然后用户得到这个压缩包后，将其解压缩后即可运行程序，Eclipse 软件就是这种方式。

另一种方式是用 InstallShield、InstallAnyWhere 这样的安装程序制作软件来创建一个单一的 setup.exe 文件，它具有向导式的安装界面，而且还可以往 windows 的程序栏插入菜单项，关于这些安装程序制作软件的具体使用请参阅相关书籍。

第 9 章　Eclipse 的 J2EE 开发

Eclipse 默认安装是没有 J2EE 开发支持的，它需要安装第三方插件，本章的主要介绍的 J2EE 开发插件是 Lomboz，主要开发环境是 Tomcat + Lomboz + Struts + Hibernate，这是当前比较流行的一种选择。其中 Tomcat 充当 WEB 服务器；Lomboz 是 J2EE 开发的工具；Struts 提供强大的 MVC 模式支持；Hibernate 替代笨重的 EJB 来充当数据库的持久层。

以上所有的工具和软件包不仅流行、功能强大、而且是免费的，是 J2EE 开发典型搭配。本章将分三个层次来渐进式的展开讲解：

Lomboz 下的纯 J2EE 开发

融合 Struts 的 J2EE 开发

融合 Struts 和 Hibernate 后的 J2EE 开发

由于篇幅有限，本章以开发环境的安装和配置为重点，并辅以一个典型而有深度的实例来演示具体的开发操作，最后给出一个扩展知识的资料索引。

本章和第 8 章一样也使用 CVS 来管理所有例程，在每一节的标题后会用括号显示这一节的版本号。本章具体的环境为：WindowsXP + JDK1.4.2_06 + Eclipse3.1M4 + cvsnt2.0.58d + Tomcat5.0.28 + Lomboz3.1.0 + Struts 1.2.4。

9.1　WEB 环境的搭建（V0010）

9.1.1　下载 CVS 版本注意事项

由于 V0010 版，存在一些空目录，而这些空目录也是必须要的，否则项目会出错。这需要修改一个 CVS 的配置，如下图 9.1 所示，打开 Eclipse 的首选项→小组→CVS→将"修剪空目录"项取消勾选。

图 9.1　修改 CVS 配置

9.1.2　Tomcat 的下载与安装

这一节先搭建好 Tomcat 环境，Tomcat 的下载安装和 Eclipse、Lomboz 都没有直接关系，它是完全独立的。

1、下载 Tomcat

（1）用IE打开Tomcat的下载页面：http://jakarta.apache.org/tomcat/index.html，选择页面左边的链接“Binaries”，转到下图 9.2 所示的页面：

图 9.2　Tomcat 项目选择

（3）单击上图中标识的“Tomcat”项，出现如下图 9.3 所示的页面

图 9.3　具体下载项

（4）下载上图 9.3 所示的“5.0.28.exe”项，下载后的文件名为：jakarta-tomcat-5.0.28.exe

注意：

（1）不要下载 Tomcat5.5.*版，因为那需要 JDK5.0 的支持；也不要下载 4.1.*版，它的功能太弱了。因为不同版本之间的安装和配置都会有所不同，为了和本教程同步，一定要下载 5.0.28 版。

（2）如果用 FlashGet 等多线程下载工具无法下载，则改用原始的 IE 右键菜单的“另存为…”项来下载。

2、安装 Tomcat

安装 Tomcat 的过程比较简单，双击得到的下载文件：jakarta-tomcat-5.0.28.exe，开始安装。

（1）选择安装组件。接受默认的勾选即可，如下图 9.4 所示。

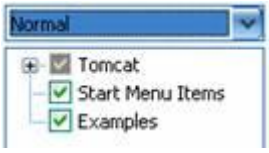


图 9.4　选择组件

（2）选择 Tomcat 安装目录。也一样接受默认值，将安装到 C:\Program Files\Apache Software Foundation\Tomcat 5.0 目录下，如下图 9.5 所示：

图 9.5　Tomcat 的安装目录

（3）选择 HTTP 监听端口（Port），如下图 9.6 所示。默认端口是 8080，如果 8080 端口已被你电脑上的其他软件所占用（如 IIS、JBoss 等），则可以另选择一个空闲的端口。最后，给 Tomcat 的超级管理员 admin 设为一个密码（本书设为 123456）。

图 9.6　设置端口和密码

（4）设置 Tomcat 使用的 JVM，本书的默认值为“C:\Program Files\Java\j2re1.4.2_06”，如下图 9.7 所示。很多资料都指出，在安装 JDK 时要设置设置 classpath、JAVA_HOME、path 等环境变量，但本书从第一章开始就从没有设置过这些环境变量，一样可以运行通畅，也许是新版的 JDK1.4.2_06 很好的解决了这些问题。从这一步也可以看到，Tomcat 已经在安装时定位好了 JVM 的位置，不必再手工设置了。

设置好 JVM 后，单击“install”按钮，开始安装。

图 9.7　定位 JVM 的位置

（5）安装完成之后，在 Windows 的“控制面板”→“管理工具”→“服务”窗口中，可以看到 Tomcat 已经注册为 windows 的一项服务，如下图 9.8 所示。请确定它是“手动”方式，这一点在开发时很重要，因为我们以后要通过 Eclipse 来启动 Tomcat。

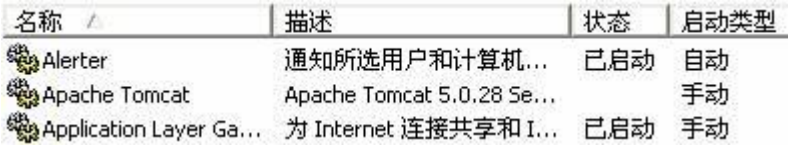


图 9.8　windows“服务”窗口

3、启动 Tomcat

虽然以后在开发时，是要通过 Eclipse 来启动 Tomcat，但现在为了测试 Tomcat 是否安装成功，暂时先启动 Tomcat。

（1）可以通过 Windows 的“开始”菜单→“Apache Tomcat5.0”组→“Configure Tomcat”项来运行 Tomcat 的配置界面（如下图 9.10 所示），这个界面包含了 Tomcat 的一些参数设置，这些设置一般都不用去改动它。直接“单击”按钮，即可启动 Tomcat。

图 9.10 Tomcat 的配置界面

（2）在 IE 浏览器中输入“http://localhost:8080”或“http://127.0.0.1:8080”，其中 8080 为安装时设置的端口号。如果启动成功，则会出现如下图 9.11 所示的页面；反之，如果没有出现此页面，则表示启动未成功，这时你需要检查前面的安装步骤是否和本书的一致。

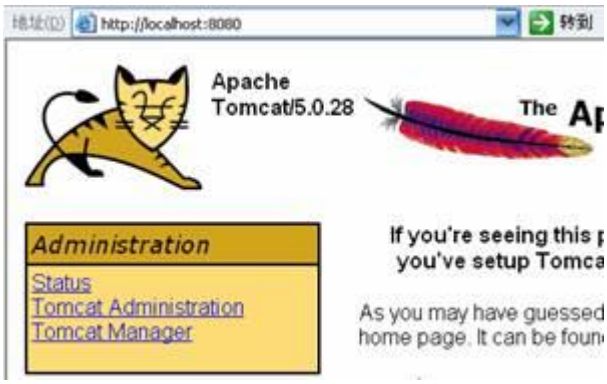


图 9.11 验证 Tomcat 是否安装及启动成功

附注：在上图页面的左部有两个链接：Tomcat Administration、Tomcat Manager，它们是用于管理 Tomcat 的，登录密码都是在安装 Tomcat 时设置的用户名 admin 和密码 123456。其中，Tomcat Adiministration 项可以设置数据库连接池、管理用户及权限、以及其他一些 Tomcat 服务器相关设置；Tomcat Manager 项主要用来发布网页管理，通过它可以轻松的将一个 WAR 包发布到 Tomcat 中。

关于 Tomcat 中文问题的解决，请参阅 9.4.6 节。

9.1.3 Lomboz 的下载与安装

下载 Lomboz 时一定要针对 Eclipse 的版本来选择相应的 Lomboz 版本下载，否则对应版本不同，很有可能会导致 Lomboz 无法正常使用。本章由于依然要使用 CVS，所以还是用 Eclipse3.1M4 版，Lomboz 选择相应的 3.1.0 版。

1、下载 Lomboz

Lomboz的下载地址是:
http://forge.objectweb.org/project/showfiles.php?group_id=97 ，下载页面如下图 9.12 所示，请选择for Eclipse3.1.x的Lomboz来下载，而且还需要同时下载emf包（如图中箭头所示）。

下载后的文件名为:

org.objectweb.lomboz_3.1.0.N20050106.zip

emf-sdo-runtime-I200412160800.zip

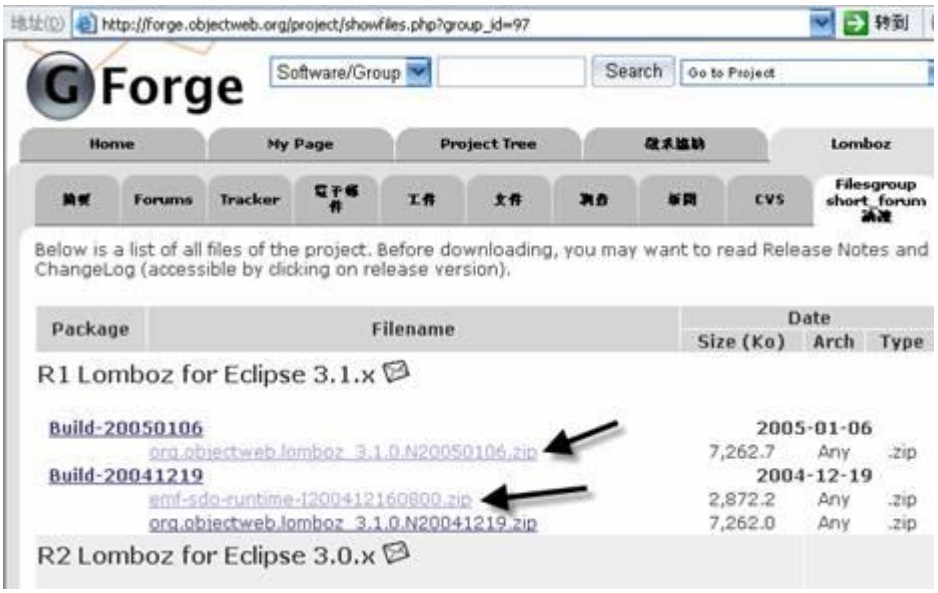


图 9.12　Lomboz 的下载页面

2、安装 Lomboz

（1）因为 Lomboz、emf 是 Eclipse 的插件，所以它和其他 Eclipse 插件的安装方法一样，本书采用 Links 式的插件安装方法，具体步骤不再重复，请参阅 1.2 节的安装步骤。

下图 9.13 是安装完成后的目录结构:

图 9.13　lomboz、emf 的安装目录结构

其中图 9.13 中的 links 目录有新创建的两个文本文件:

文件 lomboz.link，内容仅一句:

```
path=lomboz_3.1.0.N20050106
```

文件 emf.link，内容也仅一句:

```
path=emf-sdo-runtime-I200412160800
```

（2）验证 Lomboz 是否安装成功

启动 Eclipse。如果安装成功，选择"文件"→"新建"→"项目"会出现如下图 9.14 所示的 Lomboz 项目。

图 9.14　验证 Lomboz 是否安装成功

（3）如果未能出现上图画面，请做如下检查和尝试:

删除 eclipse 目录下的子目录 configuration，再启动 Eclipse 试一试。

检查 Lomboz 的版本是否和 Eclipse 的一致。

Links 文件中的 path 项是否设置正确。

Lomboz 的目录结构是否正确: ..\lomboz_3.1.0.N20050106\eclipse\plugins，注意 lomboz_3.1.0.N20050106 和 plugins 的中间还有个 elcipse 目录。

9.1.4　Lomboz 的环境设置

安装完 Lomboz 之后，还需要针对 Tomcat 做一些设置才能用于开发 WEB，具体操作步骤如下:

（1）打开 Eclipse 的首选项，设定 JDK 的 tools.jar 包的位置，本书是“C:\jdk\lib\tools.jar”，如下图 9.15 所示:

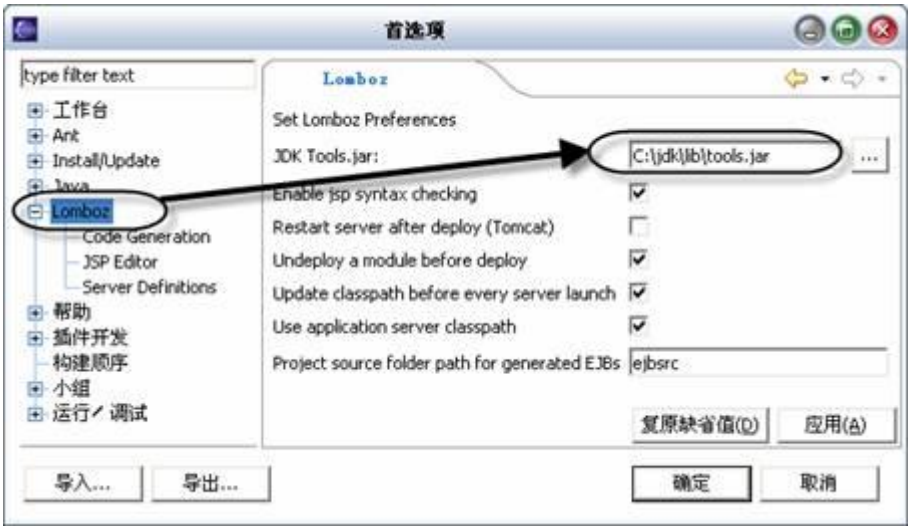


图 9.15　设定 JDK 的 tools.jar 包的位置

（2)如下图 9.16 所示,注意,在 Server types 项的下拉框中,要选择和当前所用 Tomcat 版本相对应的项; Application Server Directory 和 Classpath Variable 两项都是指向 Tomcat 的安装目录: C:\Program Files\Apache Software Foundation\Tomcat 5.0。

图 9.16　Tomcat 在 Lomboz 中的设置

（3）Tomcat5.0.28 版本在 Lomboz 中无法启动，必须还要做一些小修改。到 Lomboz 插件的“..\lomboz_3.1.0.N20050106\eclipse\plugins\com.objectlearn.jdt.j2

ee_3.0.1\servers”目录中，可以看到各种 Web 服务器的配置文件，它们都会显示在上图 9.16 的 server types 下拉框中，除了 tomcat50x.server 文件外，其他都不需要，把它们都删除掉或者备份到其他地方。最后，用记事本打开 tomcat50x.server，并将所有
“${serverRootDirectory}/bin;${serverRootDirectory}/common/endorsed”项替换成“${serverRootDirectory}/common/endorsed”，共有两处，约在文件中的 35、39 行位置。

9.1.5 JSP 的 HelloWorld

本小节将写一个 JSP 的 HelloWorld，用来验证以上 Tomcat 和 Lomboz 的环境是否安装成功。

1、设置 Java 的构建路径

打开 Eclipse 首选项，如下图 9.17 所示，选择“java”→“构建路径”→选择“文件夹”项。经过此步设置之后，新建的 Java 项目（包括 J2EE 项目）就会默认以 bin 为输出目录。

注意：这一步相当重要，因为用 Lomboz 创建 J2EE 项目时，是无法象创建普通 Java 项目那样选择“项目布局”的，此时 J2EE 项目的输出目录将会是在项目根目录下，以后 JavaBean 的 java 文件也会和 class 文件混在一块，非常不便。更关键的是，在后面会要重新定位 JavaBean 的输出路径，如果不经过这一步，则定位 JavaBean 的输出路径时，整个项目的输出路径也会自动定位到 bin 目录下，但此时项目结构都会调整，容易导致混乱。总之，此步一定不能省略。

图 9.17　设置 Java 项目的构建路径

2、创建一个 J2EE 项目

（1）重启 Eclipse。选择“文件”→“新建”→“项目”，选择如下图 9.18 所示的“Lomboz J2EE Project”项目，然后单击“下一步”。

图 9.18　选择“Lomboz J2EE Project”项目

（2）输入项目名称 myweb，然后单击“下一步”。

（3）在接下的“定义 Java 构建设置”页中不做任何改变，直接单击“下一步”。

（4）最后一个页面是 J2EE 的设置，如下图 9.19、9.20 所示。共有三步：创建一个名为 hello 的 Web Modules（WEB 模块）；在 Targeted Servers 选项卡中，选择“Apache Tomcat v5.0.x”项并单击“Add”加入；单击“完成”按钮，开始生成一个 J2EE 项目。

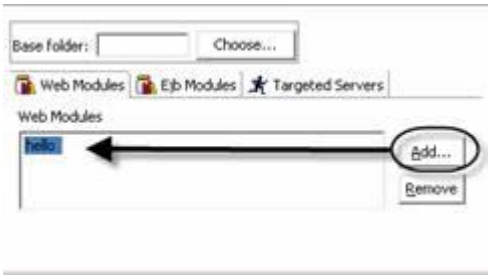


图 9.19　创建一个 Web Modules

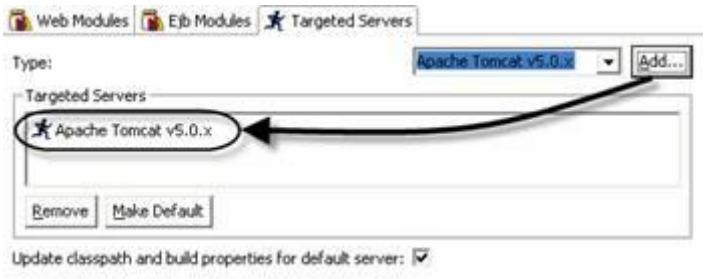


图 9.20　设置 Targeted Servers

（5）完成以上操作之后，“包资源管理器”视图中会出现如下图 9.21 所示的项目结构。

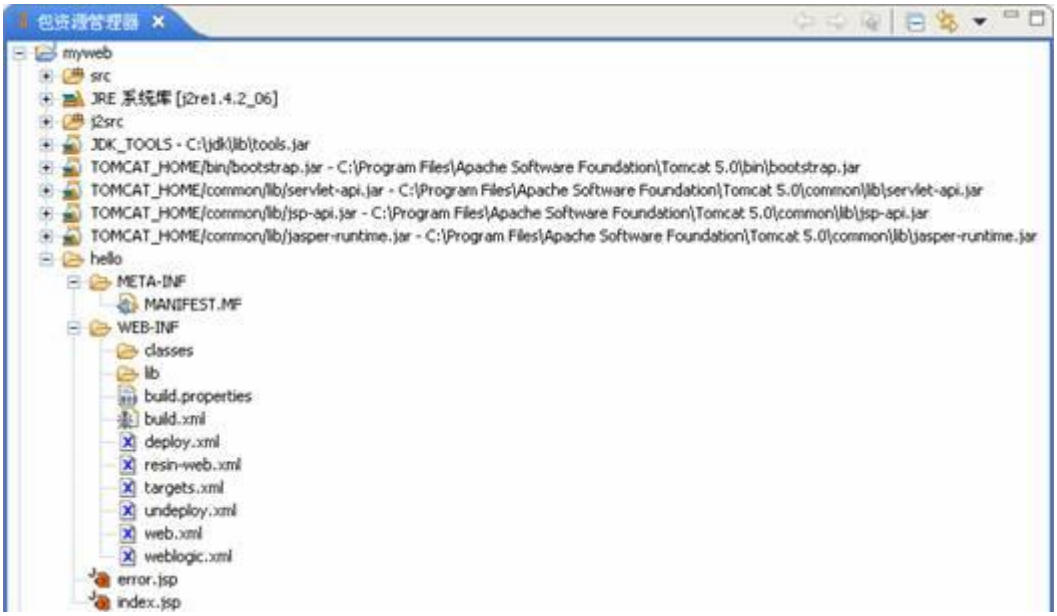


图 9.21　myweb 项目的项目结构

3、在 Lomboz 中启动 Tomcat

右键单击“hello 模块”，弹出如下图 9.22 所示的右键菜单，选择 Run Server 来启动 Tomcat（启动前确保 Tomcat 还是停止状态）。在这个菜单中还有其他常用的菜单项：

Stop Server – 停止 Tomcat

Debug Server – 用调试方式启动 Tomcat。在 WEB 开发中，它比 Run Server 更常用。

Check All JSP Syntax – 检查项目中所有 JSP 文件的语法是否符合规范

Undeploy Module – 删除已经发布在 Tomcat 上的 WEB 模块

Deploy Module – 发布 WEB 模块到 Tomcat 上

Show in Browser – 在 IE 中预览本 WEB 模块的效果。

图 9.22　hello 模块的右键菜单

如果启动 Tomcat 成功，在控制台会显示如下图 9.23 所示的字符串。

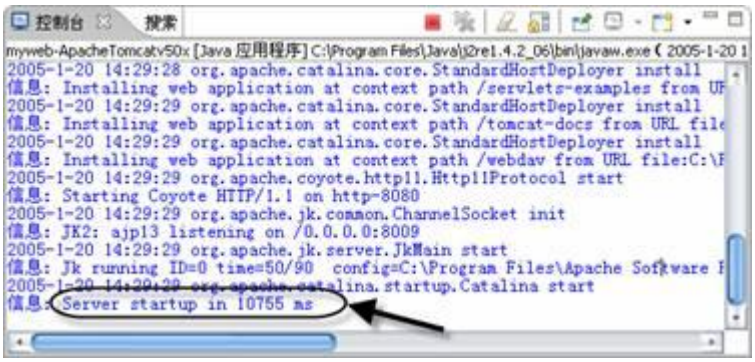


图 9.23　控制台的输出显示

4、发布 hello 模块

右键单击 hello 模块，打开如上图 9.22 所示的右键菜单，选择 Deploy Module 项，将 hello 模块发布到 Tomcat。

从下图 9.24 的控制台输出，可以看出 Lomboz 使用 Ant 来发布网页，每一行都显示出 hello 模块的打包发布过程，下面给出一些关键词解释:

mkdir - 创建目录

copy - 复制文件

jar - 用 JDK 的 jar 来打包（这里是打包成 hello.war）

delete - 删除文件

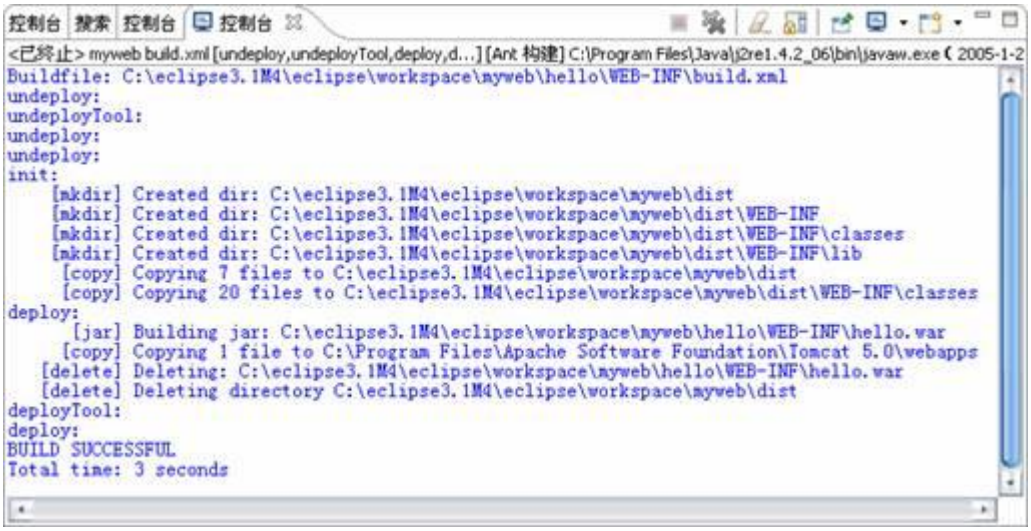


图 9.24 发布 hello 模块时的控制台输出

再次调出 hello 模块的右键菜单，选择 Show in Browser 项。Lomboz 将打开 IE 浏览器，得到如下图 9.25 所示的效果，也可以直接打开 IE 浏览器，输入地址 “http://127.0.0.1:8080/hello/”来查看效果。这个页面显示的是 index.jsp 文件。

图 9.25 用 IE 来查看网页效果

5、修改 index.jsp

如下图 9.26 所示，修改 index.jsp 来显示一个 HelloWorld 字符串。

图 9.26　修改 index.jsp

保存好之后，还要再用“Deploy Module”菜单项重新发布 hello 模块，然后才能在 IE 中看到修改后的效果。

6、一些相关问题

（1）如果看不到修改效果，有可能是 IE 的页面缓存的原因，可以尝试如下解决办法：关掉 IE，然后再打开，进入“工具”→“Internate 选项”，单击下图 9.27 中的“删除文件”按钮来删除 IE 的网页缓存。

图 9.27　删除 IE 页面缓存

（2）同样是因为缓存原因，在停止 Tomcat 服务后，即使刷新网页却依然能正常显示。将 IE 关掉重启，页面即会无法访问。

（3）如果是在 Eclipse 中启动 Tomcat 的，则关闭 Eclipse，Tomcat 服务也随之停止。但建议还是使用“Stop Server”菜单项来正常停止 Tomcat 服务。

9.1.6　如何不必发布就可以在 IE 上显示 WEB 修改效果

经过前面设置后，虽然可以开发 WEB 了，但每一次修改都要重新发布 hello 模块，才能在 IE 上显示修改后的效果，这无疑是开发时无法接受的，照这样，开发的时间进度至少要增加一倍。本小节将给出不必不发布就可以在 IE 上显示修改效果的方法。

首先，解决的办法是基于以下知识的：

在发布 hello 模块时，Lomboz 是将 hello 模块打成一个 WAR 压缩包，然后复制到 Tomcat 的 webapps 目录，在 IE 上显示的网页就是来自于这个目录下的 WAR 压缩包中，所以不能直接显示修改后的 JSP 文件也是可以理解的了。

Tomcat 要发布网页，不是必须得打成 WAR 包，也可以发布未经压缩的文件目录。实际项目中，直接发布零散文件的方式居多，因为这样更新 JSP 文件比较方便。

在 Tomcat 安装目录下的 conf 子目录里有一个名为 server.xml 的文件，它可以用来定义一个新的 WEB 应用。

由上面的知识，可以得出以下解决思路：通过修改 server.xml 文件，定义一个新的 WEB 应用，将这个 WEB 应用定位到 Eclipse 的 workspace 目录中的 myweb 项目。这样设置

以后，IE 显示的文件就是 Eclipse 中正在编写的 JSP 文件了，也就是说，不必再经过打包成 WAR 发布这一步。

具体操作步骤如下:

（1）为了避免干扰，先将原来发布的 hello 模块删除。

打开Tomcat主页面: http://127.0.0.1:8080/。选择链接“Tomcat Manager”，输入用户名密码（admin、123456），得到如下图 9.28 所示页面。单击hello模块右侧的“Undeploy”将hello模块从Tomcat上的撤消发布。

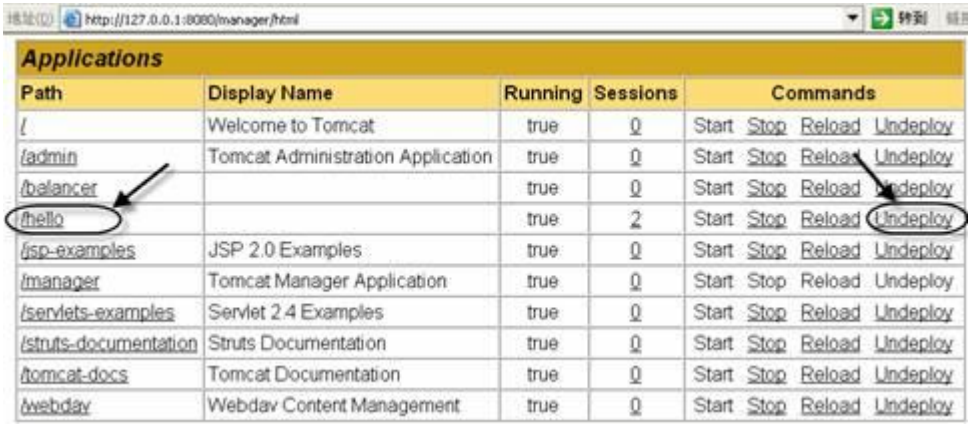


图 9.28　撤消 Tomcat 上的 hello 模块

（2）修改 server.xml，定义一个新的 WEB 应用

server.xml 此文件的具体路径如下: C:\Program Files\Apache Software Foundation\Tomcat 5.0\conf\server.xml。此 server.xml 文件最末尾的</Host>项之前插入一项<Context>的设置，<Context>的具体代码如下:

```
<Context path="/hello"
        reloadable="true"
        docBase="C:\eclipse3.1M4\eclipse\workspace\myweb\hello"
        workDir="C:\eclipse3.1M4\eclipse\workspace\myweb\bin" />
```

代码说明:

注意一定要将以上代码加在紧靠</Host>项之前，<Context>的几个属性可以分行写，也可以写成一行。

path – 是指WEB模块的名称hello，这样其访问地址为: http://127.0.0.1:8080/hello/

docBase – 定义 jsp 文件位置。本处指向 Eclipse 中 hello 模块的路径

workDir – 在 IE 显示之前，JSP 要先编译成 servlet，这个属性就是定义 hello 模块输出的 servlet 的所在位置。如下图 9.29 所示，因为所建

的 myweb 项目默认的输出路径为 myweb\bin 目录，所以这里的 workDir 也定位到此 myweb\bin 目录。

图 9.29 myweb 项目的默认输出文件夹

（4）右键单击“hello”模块→选择Lomboz J2EE→选择Debug Server（或Run Server）。然后，在IE浏览器中输入“http://127.0.0.1:8080/hello/”来查看效果。最后，随便修改一下index.jsp文件，直接刷新一下IE，如果可以看到修改后的效果，表示以上所有设置成功。

如下图 9.30 所示的“导航器”视图（注意：不是“包资源管理器”视图），index.jsp 在经过 IE 显示之后生成几个新文件和目录（可能需要先刷新一下 myweb 项目）：

图 9.30　myweb 项目结构

9.1.7　配置 Tomcat 的数据库连接池

在 WEB 开发中，有没有数据库连接池对 WEB 性能的影响非常大，Tomcat 有自带的连接池，这一节就来配置 Tomcat 的连接池。

1、复制 JDBC 连接包

将第 8 章使用的 JDBC 连接包 mysql-connector-java-3.0.16-ga-bin.jar 复制到 C:\Program Files\Apache Software Foundation\Tomcat 5.0\common\lib 目录下，common\lib 是 Tomcat 的全局引用库的所在目录，Tomcat 在启动时会自动加载这个目录中的所有 JAR 包。

有些网上的文章说也可以将数据库连接包复制到 WEB 应用的 WEB-INF\lib 中（本例的 myweb\hello\WEB-INF\lib 目录），这个目录是 hello 模块发布时会自动加载的一个包目录。但经笔者实验，如果连接包将放在此目录中，不用数据库连接池方式来访问数据库，则连接包可以正常使用；如果使用 Tomcat 连接池，则会出错误，连接包无法使用。

2、进入 Tomcat 的配置页面

用IE浏览器输入地址：http://127.0.0.1:8080/admin/ ，打开Tomcat服务器配置的登录页面，再输入用户名admin、密码 123456，进入Tomcat的配置页面，如下图 9.31 所示：

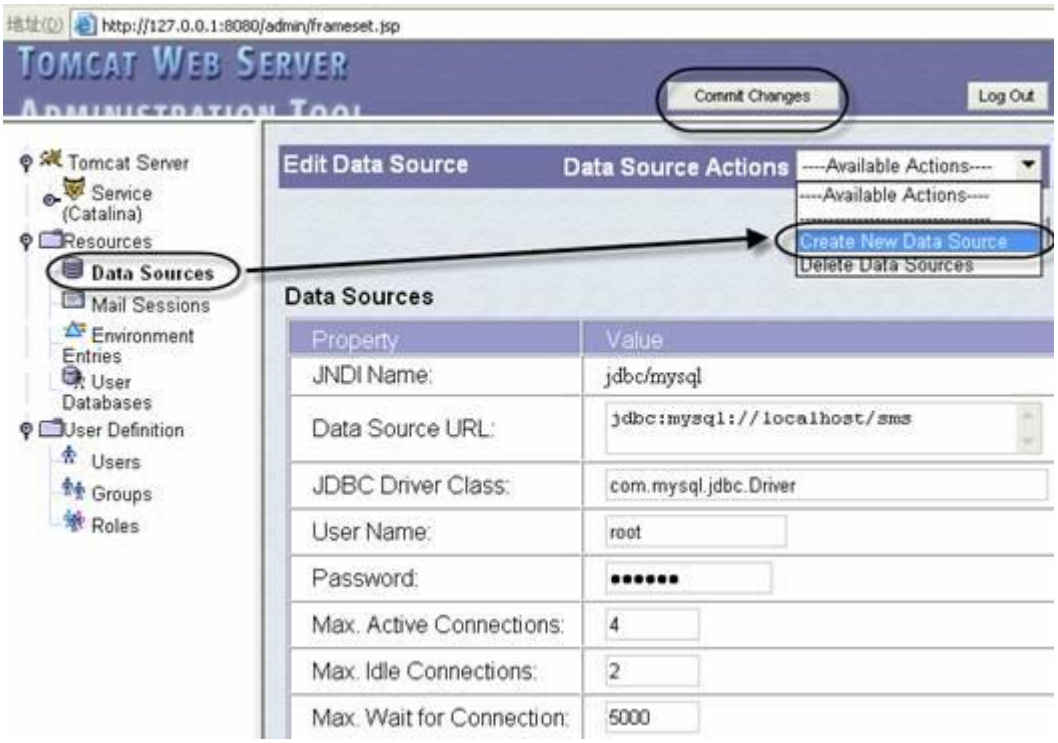


图 9.31　连接池设置

单击左边的树结点“Data Source”→选择右上角的下拉框的“Create New Data Source”项，然后在表格中输入相应的连接池配置信息：

JNDI Name: jdbc/mysql - 设置连接池的 JNDI 名，在 Java 程序会用到此名。

Data Source URL: jdbc:mysql://localhost/sms - 数据库连接字串，sms 是数据库。

JDBC Driver Class: com.mysql.jdbc.Driver - JDBC 连接类。

User Name: root - 数据库登录用户名。

Password: ****** - 数据库登录密码。本例为 123456。

Max. Active Connections: 4 - 最大连接数。实际应用时，应该根据 WEB 使用情况设置得大一些；开发时，4 个连接足够了。

Max. Idle Connections: 2 - 最大空闲连接数。

Max. Wait for Connection: 5000 - 最大等待连接限制。

Validation Query: 验证用的查询语句，可以不填。

填写完以上信息之后，单击右下角的“Save”按钮保存修改，再单击右上角的“Commit Changes”按钮提交修改。

3、修改 Tomcat 的 server.xml 文件

server.xml 文件的具体路径: C:\Program Files\Apache Software Foundation\Tomcat 5.0\conf\server.xml，在原来的<Context>项中加入一个子项< ResourceLink>:

```
<Context path="/hello"
        reloadable="true"
        docBase="C:\eclipse3.1M4\eclipse\workspace\myweb\hello"
        workDir="C:\eclipse3.1M4\eclipse\workspace\myweb\bin">
   <ResourceLink name="jdbc/mysql"
                 global="jdbc/mysql"
                 type="javax.sql.DataSourcer"/>
</Context>
```

4、测试数据库连接池

将以下测试程序命名为test.jsp，创建在hello模块的根目录下，然后通过IE地址: http://127.0.0.1:8080/hello/test.jsp来访问。这个测试程序从数据库连接池中获得一个数据库连接对象Connection，然后再查询数据库的iuser表，并用name（姓名）列的数据打印出来（注: iuser是在第 8 章创建的用户表）。test.jsp运行效果如下图 9.32 所示:

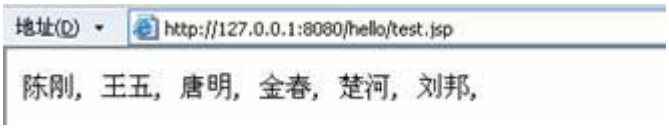


图 9.32 test.jsp 的效果

test.jsp 源代码如下:

```
<%@ page contentType="text/html; charset=GBK"%>
<%@ page import="java.sql.*"%>
<%@ page import="javax.sql.*"%>
<%@ page import="javax.naming.*"%>
```

```
<%
Connection con=null;
Statement sm=null;
ResultSet rs=null;
try{
   InitialContext ctx=new InitialContext();
   DataSource
ds=(DataSource)ctx.lookup("java:comp/env/jdbc/mysql");
   con = ds.getConnection();
   sm = con.createStatement();
   rs = sm.executeQuery("select * from iuser");
   while(rs.next())
      out.println(rs.getString("name")+",");
}catch(Exception e){
    e.printStackTrace();
}finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {}
        rs = null;
    }
    if (sm != null) {
        try {
            sm.close();
        } catch (SQLException e) {}
        sm = null;
    }
    if (con != null) {
        try {
            con.close();
        } catch (SQLException e) {}
```

```
        con = null;
    }
}
%>
```

程序说明:

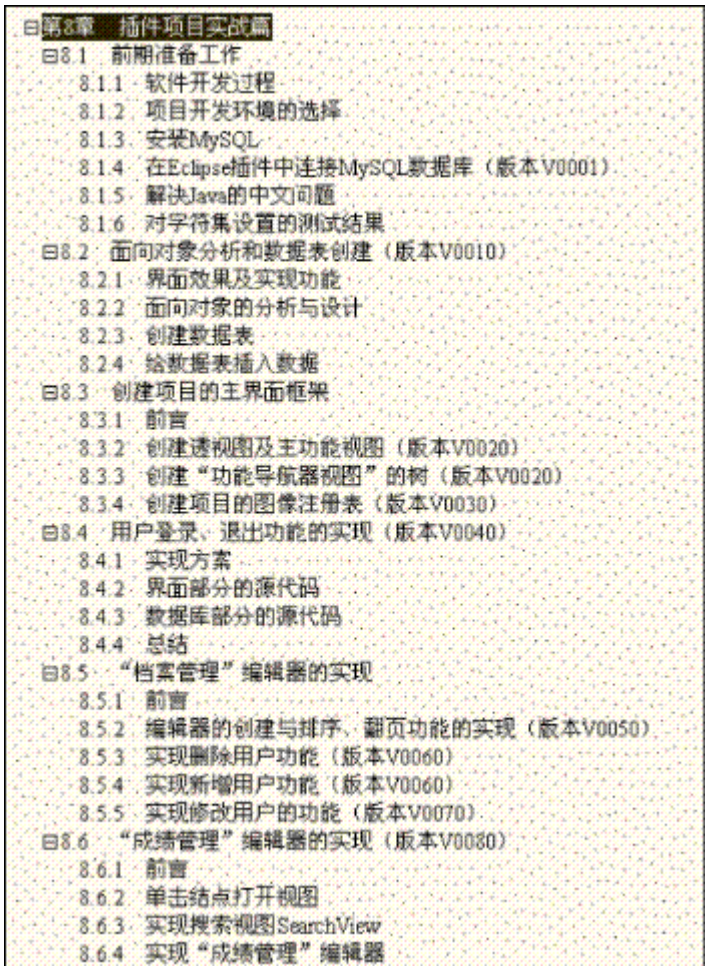
<%@ page contentType="text/html; charset=GBK"%> 这一行是设置网页的字符集，也是解决中文乱码问题的关键一句。如果是纯 html 页面，则应在</HEAD>项之前加入这样一句：<META http-equiv=Content-Type content="text/html; charset=GBK">。

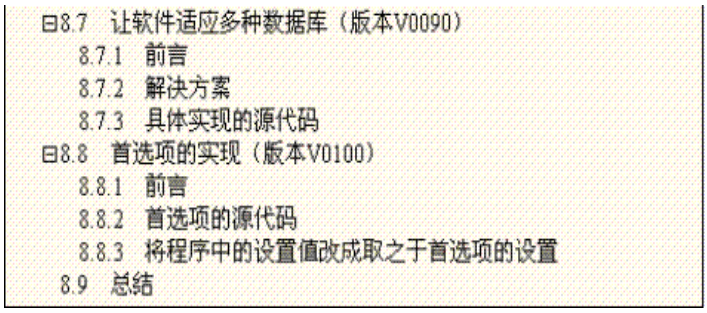
<%@ page import="java.sql.*"%> 这一句类似于 Java 中的 import java.sql.*。

ctx.lookup("java:comp/env/jdbc/mysql"); 这一句中 comp/env 是固定不变的，jdbc/mysql 是前面连接池设置的 JNDI Name。

在程序最后一定要关闭数据库连接（实际是将连接返回给连接池，并没有真正关闭），否则，很快就会因连接数耗尽，而无法再正常显示 JSP 页面。

今天再帖出在“插件项目实战”一章中关于建模的。内容虽然简单，但其中的方法我认为还是很重要的，因为在浏览很多帖子发现在建模时，还是有不少争论的，我估计至少有 70%的 Java 程序员，无法很好的做到面向对象设计和分析，本节多少也反映了我的一些经验和观点吧，希望对大家有所帮助。





8.2 面向对象分析和数据表创建（版本V0010）

8.2.1 界面效果及实现功能

本章项目是编写一个学生成绩管理软件，由于主要目的是给出一个项目开发的示例，所以这个软件的功能是做得相当简单的，也不存在需求分析阶段。关于学生成绩管理软件的一些功能及概念，也不再多加说明，毕竟大家都是从学校和考试里走出来的，对这些已经很熟悉了。

本章项目的主体界面框架如下图 8.19 所示:

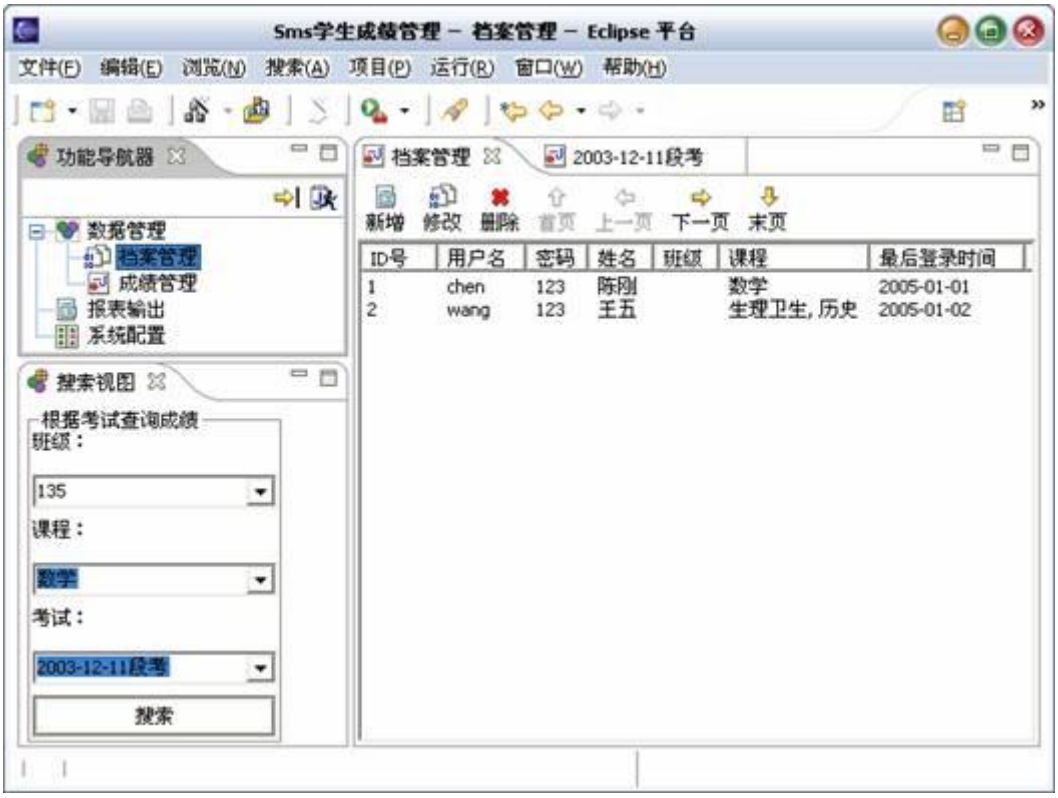


图8.19　主体界面框架

功能说明:

左上部是主功能导航器视图（简称为主功能导航器或主功能视图），其中提供了一个功能结点树，本章将实现"档案管理"和"成绩管理"两个结点的功能。

右部是一个编辑器，当单击"档案管理"结点时将生成一个编辑器。

左下部是成绩管理的搜索视图，可以根据这个视图设置的搜索条件，查询出相应的考试成绩。

右部还有一个名为"2003-12-11 段考"的编辑器，当单击左下部的"搜索"按钮时将生成此编辑器，如下图 8.20 所示:

图 8.20　成绩编辑器

8.2.2　面向对象的分析与设计

面向对象的分析与设计，也称 OOAD（Object Oriented Analyse Design）。因为它能够更准确自然的用软件语言来描述现实事物，并使得在它基础上构建的软件具有更好的复用率、扩展性及可维护性，所以 OOAD 是当前最重要的软件方法学之一。

OOAD 和 Rose、Together 等 UML 软件没有必然的关系，OOAD 是一种方法，UML 是描述这种方法的图形语言，而 Rose 等则是使用 UML 的具体工具。OOAD 的关键在于思维方式的转变，而不是工具的使用，即使只用铅笔和白纸也可以成为一个优秀 OOAD 专家。

现在大学的课程以 C、Basic、VB、FoxPro 居多，即使是用 C++、Java，也是可以用面向过程的方式来编写程序，所以使用面向对象的语言并不代表你是以面向对象的方式来思考和编程。徒具对象的形，而无对象的神，是现在一般程序员的最大缺陷所在。
以本项目为例，大多数习惯于面向过程的编程思维方式的开发人员，一般在做完需求分析后，便开始设计数据库的表结构，而在编码阶段才开始考虑根据表结构来进行对象的设计与创建，这种开发方式就是带有过去很深的面向过程、面向数据库表编程的烙印。
所谓“万物皆对象”，OOAD 应该是把对象做为思考的核心，而不是仅仅把“对象”当成一种编程的手段，应当先完成对象设计，然后再根据对象创建表，这是最基本的次序。
当然这种方式在转化成数据库时会遇到一些困难和阻力，毕竟数据库不是面向对象的，SQL 语言也不是面向对象的。但 Hibernate、JDO、EJB 等数据库持久化技术，已经可以让开发者用完全的面向对象方式来编程，而不必忍受“对象”到“关系”转化的痛苦。
为了让读者可以了解如何手工完成“对象”到“关系”的转化，本插件项目仍然使用纯 JDBC 方式来实现。在第 9 章会讲解 Hibernate 的使用，所谓“先苦后甜”，通过两种方式的比较，读者能更深的体会 Hibernate 等数据库持久化技术的美妙之处。
本章的学生成绩管理软件有以下对象：学生、老师、年级、班级、课程、成绩、考试，本项目所有对象创建在 cn.com.chengang.sms.model 包下，如下图 8.21 所示。接下来会具体分析一下这些对象，并给出其源代码和 UML 类图。

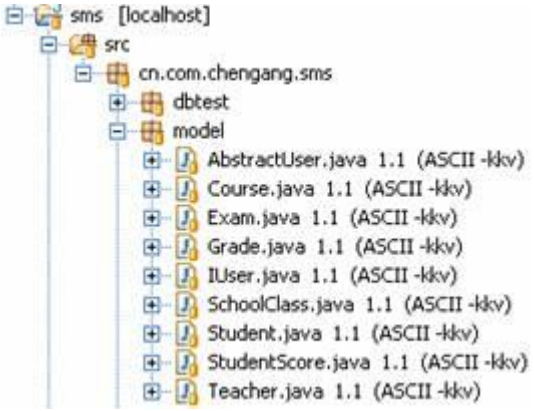


图8.21　数据对象所在的包

1、用户对象：学生、老师

这个系统有可能会存在一个前台网站，比如：老师用 Eclipse 做客户端来管理成绩，而学生则通过一个网页来查询成绩，所有的数据集中在学校的中心服务器上。因此系统的用户有两种：学生、老师，这两种用户有一些信息是相同的，有些则不同。比如他们都有用户名、姓名、密码等，而学生没有老师的课程属性，老师则没有学生的班级属性。
由上面的分析，我们将两种用户的共性抽象成一个接口：IUser，这个接口有如下属性：数据库 ID 号（Id）、用户名（userId）、密码（password）、姓名（name）、最后登录时间（latestOnline）。另外，学生类（Student）有班级属性（SchoolClass），老师类（Teacher）则有课程（Course）属性，学生类和老师类都实现于 IUser 接口。
将用户抽象成一个接口的另一个好处就是：使用户类置于同一个规范之下。今后要新增加一个种类型的用户，比如：家长用户，只需要再实现 IUser 接口即可。“接口”是用 Java 进行 OOAD 开发的一个最重要的概念，也是成为一个优秀的 Java 设计师所必须掌握和熟练使用的概念。

其他说明：类的实例变量有多种叫法：通用的名称是“实例变量”或“属性”；在实体类中因为和数据表的字段相对应，也可称之为“字段”；有些书籍文章也称之为“域”。

先给出用户类的 UML 设计图，如下图 8.22 所示：

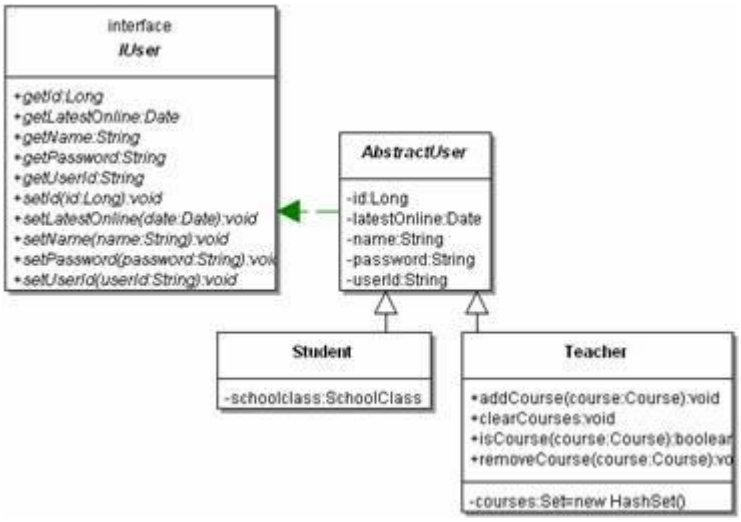


图8.22　用户类的UML类图

用户类的源代码如下:

（1）用户接口 IUser

```
package cn.com.chengang.sms.model;

import java.util.Date;

public interface IUser {

    /**

     * 得到数据库 ID

     */

    public Long getId();

    /**

     * 设置数据库 ID

     */

    public void setId(Long id);

    /**

     * 得到用户名

     */

    public String getUserId();

    /**

     * 设置用户名
```

```
 */
public void setUserId(String userId);
/**
 * 得到密码
 */
public String getPassword();
/**
 * 设置密码
 */
public void setPassword(String password);
/**
 * 得到用户姓名
 */
public String getName();
/**
 * 设置用户姓名
 */
public void setName(String name);
/**
 * 得到最后登录时间
 */
public Date getLatestOnline();
/**
 * 设置最后登录时间
 */
public void setLatestOnline(Date date);
```

}

程序说明:

接口规定只能定义方法，不能定义属性变量，所以本例只定义了用户各属性的 set/get 方法。

接口定义的方法前面是否有 public 或 abstract 都是一样的，本例加了 public，你也可以去除，两者效果相同。

这里需要注意的是 Date 对象是 java.util.Date，不要和 java.sql.Date 混淆。

（2）实现接口 IUser 的抽象类 AbstractUser

每一个具体用户类（学生、老师）都要实现一遍接口 IUser 中定义的方法，而这些方法的代码都是一样的，所以我们用一个抽象类 AbstractUser 来统一实现 IUser 接口中的公共属性，我们把这种抽象类称之为"默认实现抽象类"。AbstractUser 不仅提供了方法的实现，也提供了属性变量的定义，所有的用户子类都将继承并拥有这些属性。

AbstractUser 类的具体代码如下:

```
package cn.com.chengang.sms.model;

import java.util.Date;

abstract class AbstractUser implements IUser {

    private Long id; //数据库 ID

    private String userId; //用户名

    private String password; //密码

    private String name; //姓名

    private Date latestOnline;//最后登录时间


    /********以下为接口 IUser 的实现方法***********/

    public Long getId() {

        return id;

    }


    public void setId(Long id) {

        this.id = id;

    }
```

```
public String getUserId() {
    return userId;
}

public void setUserId(String userId) {
    this.userId = userId;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public Date getLatestOnline() {
    return latestOnline;
}
```

```
    public void setLatestOnline(Date latestOnline) {
        this.latestOnline = latestOnline;
    }
}
```

（3）学生类 Student

学生类 Student 继承自抽象类 AbstractUser，所以也拥有了抽象类中所有的属性和方法，因此这里只需定义学生类独有的属性和方法。

```
package cn.com.chengang.sms.model;

public class Student extends AbstractUser {
    //学生所属班级，为了避免和类（class）的名称混淆，将其命名为 SchoolClass
    private SchoolClass schoolclass;
    /**
     * 得到学生所在班级
     */
    public SchoolClass getSchoolclass() {
        return schoolclass;
    }
    /**
     *  设置学生所在班级
     */
    public void setSchoolclass(SchoolClass schoolclass) {
        this.schoolclass = schoolclass;
    }
}
```

（4）老师类 Teacher

```
package cn.com.chengang.sms.model;

import java.util.HashSet;

import java.util.Set;
```

```
public class Teacher extends AbstractUser {
    private Set courses = new HashSet(); //所教课程
    /**
     * 得到所有课程
     */
    public Set getCourses() {
        return courses;
    }
    /**
     * 设置一批课程
     */
    public void setCourses(Set courses) {
        this.courses = courses;
    }
    /**
     * 增加一个课程
     */
    public void addCourse(Course course) {
        courses.add(course);
    }
    /**
     * 删除一个课程
     */
    public void removeCourse(Course course) {
        courses.remove(course);
    }
    /**
```

```
     * 清除所有课程
     */
    public void clearCourses() {
        courses.clear();
    }
    /**
     * 该老师是否教这个课
     */
    public boolean isCourse(Course course) {
        return courses.contains(course);
    }
}
```

程序说明:

我们将课程也看作是一种对象,命名为 Course,在后面将会给出它的代码。老师和课程是多对多的关系:一个老师有可能教多门课程,一门课程也可能有几个老师来教。当一个对象对应多个对象的情况时,比如老师,就需要一个 Java 集合(Collection)来存放这些课程,集合中的一个元素就是一门课程。

在 List 和 Set 两种集合中,本例选择了 Set 型集合。Set 的特性是其包含的元素不会重复(如果加入重复的元素也不会出错,等于没有加),但 Set 中的元素是无序排列的,如果先加入"语文"后加入"数学",以后取出显示时未必"语文"会在"数学"之前。List 型集合则不同,它按加入的先后顺序排列,而且允许加入重复的元素。

Set 是一个接口,它实际使用的类是 HashSet,在定义对象时应尽量使用效宽泛的类型,以便拥有更好的扩展性。

老师类的课程属性在 set/get 方法的基础上再加了三个方法:增加课程、删除课程、判断此老师是否教授某课程,加入这些方法主要是为了今后使用方便。

因为在类的 isCourse、clearCourses、addCourse 等方法中,当 courses 为空时都会出错,所以为了方便,在定义 courses 属性时,马上赋了一个 HashSet 值给它。

2、课程(Course)、班级(SchoolClass)、年级(Grade)对象

这三个对象比较简单。其源代码如下:

(1)课程类 Course

```
package cn.com.chengang.sms.model;

public class Course {
```

```
    private Long id;
    private String name; //课程名: 数学、语文

    public Course() {}
    public Course(Long id, String name) {
        this.id = id;
        this.name = name;
    }

    /*********属性相应的 set/get 方法*************/
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

程序说明:

对于课程这种记录条数很少的属性，似乎没有必要用 Long 型，但为了整体上的统一，因此所有对象的 id 都用 Long 类型。

这里为了在创建对象时方便，新增加了一个构造函数 Course(Long id, String name) 。

（2）班级类 SchoolClass

```
package cn.com.chengang.sms.model;

public class SchoolClass {

    private Long id;

    private String name; //班级: 43 班、52 班

    private Grade grade; //该班级所属年级


    public SchoolClass() {}

    public SchoolClass(Long id, String name, Grade grade) {

        this.id = id;

        this.name = name;

        this.grade = grade;

    }


    /*********属性相应的 set/get 方法*************/

    public Long getId() {

        return id;

    }


    public void setId(Long id) {

        this.id = id;

    }


    public String getName() {
```

```
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Grade getGrade() {
        return grade;
    }

    public void setGrade(Grade grade) {
        this.grade = grade;
    }
}
```

（3）年级类 Grade

```
package cn.com.chengang.sms.model;

public class Grade {
    private Long id;
    private String name; //年级名: 大一、初三

    public Grade() {}
    public Grade(Long id, String name) {
        this.id = id;
        this.name = name;
    }
```

```
    /*********属性相应的 set/get 方法*************/
    public Grade(int id, String name) {
        this.id = new Long(id);
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

（4）三个类的 UML 图，如下图 8.23 所示:

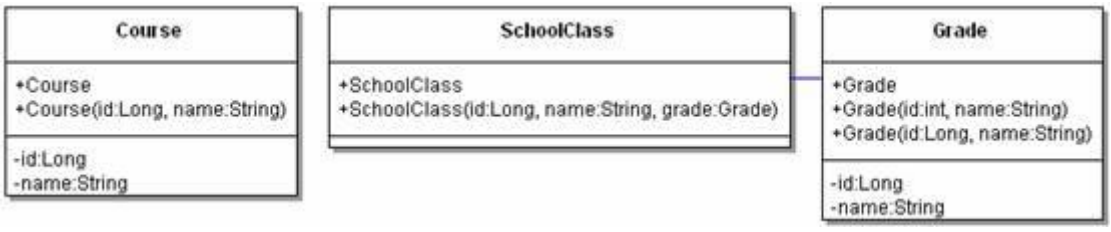


图8.23　课程、班级、年级的UML图

3、学生成绩（StudentScore）、考试（Exam）对象

学生的成绩一般要包含如下信息：是哪位学生的成绩、是哪一次考试、这位学生的得分是多少等。在这里我们将考试的信息抽取出来单独构成一个考试（Exam）对象。

学生成绩的属性有：学生对象、考试对象、分数。

学生对象前面已经给出了，分数是一个实数。

而考试对象包含如下属性：考试名称、监考老师、考试的课程、考试的班级、考试时间。如果有必要，还可以加入更多的属性字段，如：考试人数、及格人数、作弊人数等。

（1）学生成绩类 StudentScore

```
package cn.com.chengang.sms.model;

public class StudentScore {

    private Long id;

    private Exam exam; //考试实体

    private Student student; //学生

    private float score; //得分



    /*********属性相应的 set/get 方法*************/

    public Long getId() {

        return id;

    }



    public void setId(Long id) {

        this.id = id;

    }



    public float getScore() {

        return score;

    }
```

```
    public void setScore(float score) {
        this.score = score;
    }

    public Student getStudent() {
        return student;
    }

    public void setStudent(Student student) {
        this.student = student;
    }

    public Exam getExam() {
        return exam;
    }

    public void setExam(Exam exam) {
        this.exam = exam;
    }
}
```

（2）考试类 Exam

```
package cn.com.chengang.sms.model;

import java.util.Date;

public class Exam {
    private Long id;
    private String name; //考试名称，如：2004 上半学期 143 班期末语文考试
```

```
private Teacher teacher; //监考老师

private Course course; //考试的课程

private SchoolClass schoolClass;//考试班级

private Date date; //考试时间


/*********属性相应的 set/get 方法*************/

public Course getCourse() {

    return course;

}


public void setCourse(Course course) {

    this.course = course;

}


public Long getId() {

    return id;

}


public void setId(Long id) {

    this.id = id;

}


public String getName() {

    return name;

}


public void setName(String name) {
```

```
        this.name = name;
    }

    public SchoolClass getSchoolClass() {
        return schoolClass;
    }

    public void setSchoolClass(SchoolClass schoolClass) {
        this.schoolClass = schoolClass;
    }

    public Teacher getTeacher() {
        return teacher;
    }

    public void setTeacher(Teacher teacher) {
        this.teacher = teacher;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```

（3）两类的 UML 图，如下图 8.24 所示

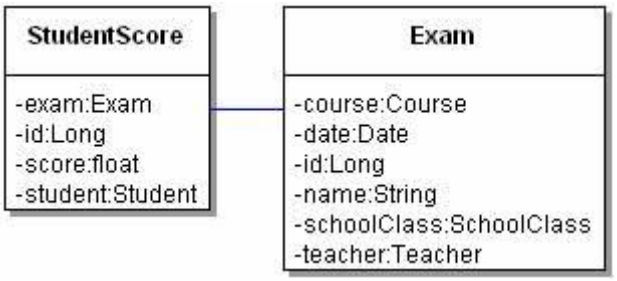


图8.24　学生成绩、考试的类图

4、总结

在年级、班级等对象设计的时候，还有一种可能的做法是——取消这些对象，并在学生类中直接使用字符型的年级、班级属性。这种方式在编程上似乎要方便一些，但不符合数据库的设计规范，它主要有以下缺点：

数据冗余 - 如果还需要增加一个"班主任"的属性，则本书的做法只需在班级类中再加一个属性，而后一种做法则需要在学生类中再加入一个班主任的属性。一个班有数十个学生，他们的老师都是一样的，这样就产生了大量的数据冗余。

修改不方便 - 如果要更改班级的名称，则本书的做法只需要修改班级表中的一条记录，而后一种做法则要更新学生表中所有的班级字段。

一致性差 - 后一种做法有可能存在一致性问题，比如某个班级也许会在学生表中存在多种名称：43、43 班、高 43 班等等。

实践建议：

在设计对象时，应该保持对象的细粒度。比如：成绩对象、考试对象的设计就是遵循这个原则。可能有些人会将考试对象取消，而将其属性合并到成绩对象中，这样做是不对的，并且以后也会造成数据表的数据冗余。

尽量为每个实体对象（表），增加一个和业务逻辑没有关系的标识属性（字段），例如本例中的自动递增属性（字段）id。在速度和可扩展性之间平衡后，建议将它定义成 java.lang.Long 类型。

设计数据库尽量依照数据库设计范式来做，不要为了书写 SQL 语句方便，而将同一字段放在多个表中，除非你对查询速度的要求极高。而且要知道这样做会导致今后数据库维护和扩展的困难，并且在更新数据时将需要更新多个表，一样增加了复杂度。

实体对象是一种纯数据对象，和数据库表有着一定程度上的对应关系，但又不是完全对应。切记不要在实体对象中加入业务逻辑或从数据库里取数据的方法，应该让其与业务逻辑的完全分离，保证实体对象做为纯数据对象的纯洁性，这样可以让它具有更高的复用性。

其他说明：本节创建的对象称之为实体对象，它是由 EJB 中的 EntityBean 提出的概念，本文采用实体对象（实体类）的称法。也可称 POJO（Plain Old Java Object，简单原始的 Java 对象），在 Hibernate 中使用 POJO 的称法较多。

21.3　用 Ant 来打包

Eclipse 内置了 Ant。Ant 是一种类似于批处理程序的软件包，它主要繁琐的工作是编写和调试自动处理脚本（一个 XML 文件），但只要有了这个脚本，我们就可以一键完成所有的设定工作。

本节还是以 myswt 这个应用程序项目的打包为例，用 Ant 来完成“编译－＞打成 JAR 包－＞复制项目引用库－＞复制本地化文件 swt-win32-3063.dll－＞输出 API 文档”这五步。

1、在 myswt 项目根目录下，创建最主要的 build.xml 文件

```
<?xml version="1.0"?>

<project name="myswt project" default="api_doc">

    <!-- 定义目录变量 -->

    <property name="src.dir" value="src" />

    <property name="bin.dir" value="bin" />

    <property name="eclipse_plugins.dir" value="c:/eclipse/plugins"
/>

    <property name="dist.dir" value="d:/dist" />

    <property name="doc.dir" value="${dist.dir}/api" />

    <property name="swt.dll" value="swt-win32-3063.dll" />

    <!-- 定义编译文件时所引用的库 -->

    <path id="master-classpath">

        <fileset dir="${eclipse_plugins.dir}" id="project_lib">

            <include
name="org.eclipse.ui.workbench_3.0.1/workbench.jar"/>

            <include
name="org.eclipse.swt.win32_3.0.1/ws/win32/swt.jar"/>

            <include name="org.eclipse.jface_3.0.0/jface.jar"/>

            <include name="org.eclipse.osgi_3.0.1/osgi.jar"/>

            <include name="org.eclipse.osgi_3.0.1/core.jar"/>

            <include name="org.eclipse.osgi_3.0.1/resolver.jar"/>

            <include
name="org.eclipse.osgi_3.0.1/defaultAdaptor.jar"/>

            <include
name="org.eclipse.osgi_3.0.1/eclipseAdaptor.jar"/>

            <include name="org.eclipse.osgi_3.0.1/console.jar"/>

            <include
name="org.eclipse.core.runtime_3.0.1/runtime.jar"/>

            <include
name="org.eclipse.jface.text_3.0.1/jfacetext.jar"/>
```

```
            <include
name="org.eclipse.ui.workbench.compatibility_3.0.0/compatibility.
jar"/>

        </fileset>

    </path>



    <!-- 首任务（空）-->

    <target name="init"/>

    <!-- 编译 -->

    <target name="compile" depends="init">

        <delete dir="${bin.dir}"/>

        <mkdir dir="${bin.dir}"/>

        <!--编译源程序-->

        <javac srcdir="${src.dir}" destdir="${bin.dir}"
target="1.4">

            <classpath refid="master-classpath"/>

        </javac>

        <!--复制图标目录-->

        <mkdir dir="${bin.dir}/icons"/>

        <copy todir="${bin.dir}/icons">

            <fileset dir="icons"/>

        </copy>

    </target>

    <!-- 打包 -->

    <target name="pack" depends="compile">

        <!-- bin 目录压缩成 JAR 包 -->

        <delete dir="${dist.dir}"/>

        <mkdir dir="${dist.dir}" />

        <jar basedir="${bin.dir}" destfile="${dist.dir}/myswt.jar"
manifest="ant_manifes.txt">

            <exclude name="**/*Test.*" />

            <exclude name="**/Test*.*" />

        </jar>
```

```
        <!-- 复制用到的库 -->
        <mkdir dir="${dist.dir}/lib" />
        <copy todir="${dist.dir}/lib">
            <fileset refid="project_lib"/>
        </copy>
        <!-- 复制本地化文件 -->
        <copy todir="${dist.dir}" file="${swt.dll}"/>
    </target>
    <!-- 输出 api 文档 -->
    <target name="api_doc" depends="pack">
        <delete dir="${doc.dir}"/>
        <mkdir dir="${doc.dir}" />
        <javadoc destdir="${doc.dir}" author="true" version="true"
use="true" windowtitle="MySWT API">
            <packageset dir="${src.dir}" defaultexcludes="yes"/>
            <doctitle><![CDATA[<h1>MySWT
Project</h1>]]></doctitle>
            <bottom><![CDATA[<i>Document by ChenGang
2005.</i>]]></bottom>
        </javadoc>
    </target>
</project>
```

代码说明:

（1）property 项是定义变量，比如<property name="swt.dll" value="swt-win32-3063.dll" />，就是定义一个变量: swt.dll=swt-win32-3063.dll。以后用这个变量则是这样: ${swt.dll}。

一般尽量将今后可能会变动的目录、文件等定义成变量，以方便维护。不象 Java 变量有类型的区分，Ant 变量是不区别目录、文件等的，所以为了见名知意，在取变量名时，目录都加“dir”后缀，这个后缀是可以任取名的。

下面给出本例用到的变量的含义:

src.dir - Java 源文件路径。value="src"的 src 是一个相对路径，它相对的是 build.xml 的所在目录位置（即项目根目录）。

bin.dir - Java 编译文件的输出路径

eclipse_plugins.dir - eclipse 的 plugins 目录

dist.dir - 打包文件的存放目录

doc.dir - API 文档的存放目录，这里用到了 dist.dir 变量，直接写 value="d:/dist/api"也未尝不可。

swt.dll - SWT 本地化文件。

（2）<path id="master-classpath">，定义编译文件时所引用的库，相当于 classpath。<fileset>项表示一个文件集，再深入一层的<include>项，则表示此文件集下的文件，它们的路径定位相对于<fileset>的 dir 属性。<fileset>还有一个 id 属性，在后面复制引用库时会用到。

也许有读者会问："你是怎么知道要引用这些文件的？"回答：看项目根目录下的".classpath"文件，就可以知道本项目要引用那些库了。实际上笔者是把.classpath 复制一份后，然后用 Editplus 编辑而得。

（3）接下来开始定义一些任务。首任务一般都让它为空（没有具体任务内容）：<target name="init"/>。

（4）Ant 中的任务有着相互的依赖（depends）关系，这些依赖关系是通过 depends 属性来定义的。当要执行一个任务时，Ant 先去执行这个任务的 depends 任务，……，Ant 就这样一直往回找下去。比如：在本例的第二行 default="api_doc"，它定义了缺省任务是 api_doc（输出 api 文档）->此任务的 depends = pack（打包）->pack 的 depends = compile（编译）->compile 的 depends=init（首任务），init 没有 depends。于是，Ant 就从 init 开始依次往回执行任务：init->compile->pack->api_doc。

如果你不想"输出 api 文档"，则将第二行的缺省任务定义成 default="pack"即可，这时整个任务链就抛开了 api_doc。

（5）<delete dir="${bin.dir}"/>删除目录。<mkdir dir="${bin.dir}"/>新建目录

（6）编译源程序，如下

```
<javac srcdir="${src.dir}" destdir="${bin.dir}" target="1.4">
    <classpath refid="master-classpath"/>
</javac>
```

srcdir - 源文件目录，其子目录中的源文件也会被 javac.exe 编译。

destdir - 编译文件输出目录。

target - 以 JDK1.4 为编译目标。

classpath - 编译的 classpath 设置，refid 是指引用前面设定的 master-classpath。

（7）将 icons（即 myswt/icons）目录的文件，复制到 myswt/bin/icons 目录中，如下:

```
<copy todir="${bin.dir}/icons">
    <fileset dir="icons"/>
</copy>
```

（8）将文件打成 JAR 包

```
<jar basedir="${bin.dir}" destfile="${dist.dir}/myswt.jar"
manifest="ant_manifes.txt">
    <exclude name="**/*Test.*" />
    <exclude name="**/Test*.*" />
</jar>
```

basedir - 源目录。

destfile - 目标目录和打成 JAR 包名。

manifest - 打包清单文件（后面给出其内容）。

exclude - 使用了通配符将某一些文件排除不打包（主要是一些测试文件）。

（9）如下，将 project_lib 的文件复制到 d:/dist/lib 目录中。project_lib 是前面“定义编译文件时所引用的库”中的文件集的 id。结果参数下图 21.25

```
<copy todir="${dist.dir}/lib">
    <fileset refid="project_lib"/>
</copy>
```

（10）将本地化文件复制到 d:/dist 目录中，如下:

```
<copy todir="${dist.dir}" file="${swt.dll}"/>
```

（11）输出 API 文档（结果参数下图 21.26）

```
<javadoc destdir="${doc.dir}" author="true" version="true"
use="true" windowtitle="MySWT API">
    <packageset dir="${src.dir}" defaultexcludes="yes"/>
    <doctitle><![CDATA[<h1>MySWT Project</h1>]]></doctitle>
    <bottom><![CDATA[<i>Document by ChenGang 2005.</i>]]></bottom>
</javadoc>
```

destdir - 目标路径 d:/dist/api

packageset – 源文件目录

doctitle – 标题

bottom – 标尾。

2、创建打包清单

为了避免和原来的 manifes.txt 同名，在项目根目录建立一个名为 ant_manifes.txt 的文件。这个文件内容中最长的是 Class-Path 项，没有必要一个个字符的敲入，它可以由项目根目录下的“.classpath”编辑而得。

ant_manifes.txt 内容如下:

```
Manifest-Version: 1.0

Main-Class: jface.dialog.wizard.WizardDialog1

Class-Path: ./lib/org.eclipse.ui.workbench_3.0.1/workbench.jar ./
lib/org.eclipse.swt.win32_3.0.1/ws/win32/swt.jar
./lib/org.eclipse.jface_3.0.0/jface.jar ./lib/org.eclipse.osgi_3.
0.1/osgi.jar ./lib/org.eclipse.osgi_
3.0.1/core.jar ./lib/org.eclipse.osgi_3.0.1/resolver.jar ./lib/or
g.eclipse.osgi_3.0.1/defaultAdaptor.ja
r ./lib/org.eclipse.osgi_3.0.1/eclipseAdaptor.jar ./lib/org.eclip
se.osgi_3.0.1/console.jar ./lib/org.ecl
ipse.core.runtime_3.0.1/runtime.jar ./lib/org.eclipse.jface.text_
3.0.1/jfacetext.jar ./lib/org.eclipse.u
i.workbench.compatibility_3.0.0/compatibility.jar
```

3、如下图 21.23 所示，选择“Ant 构建”来运行 Ant。

图 21.23　运行“Ant 构建”

运行“Ant 构建”后的结果如下图 21.23－26 所示。

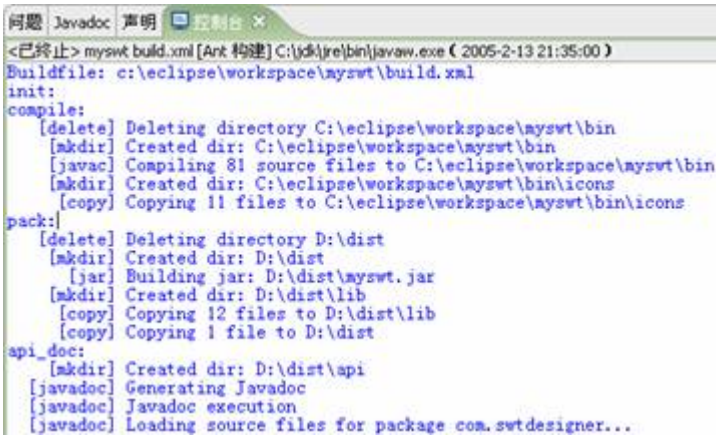


图 21.24　控制台的输出

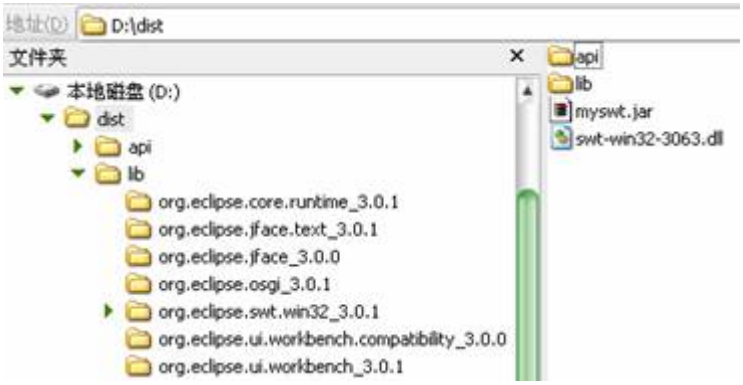


图 21.25　输出文件的目录结构图

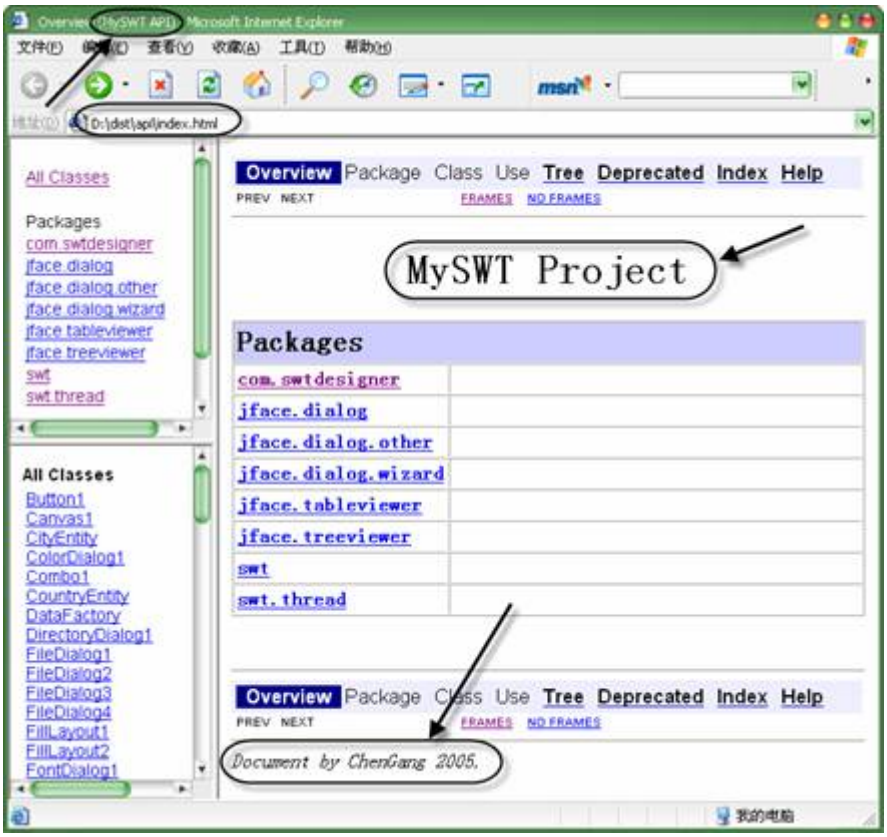


图 21.26　输出的 API 文档效果图

4、运行打包结果

除了清单文件 MANIFEST.MF 之外，myswt.jar 文件和 21.1 节所得的 myswt.jar 一样。本节没有创建 run.bat 批处理文件，而是用下图 21.27 所示的“右击 myswt.jar－＞打开方式－＞javaw”的方式来运行 myswt.jar。

图 21.27　运行 myswt.jar