

第18章 常用插件扩展点

在上一章对plugin.xml做了少量介绍，plugin.xml是插件和Eclipse内核的接口，Eclipse就象一所大宅子，它的外墙（plugin.xml）有很多的门（扩展点），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上一章建立的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用的透视图，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：

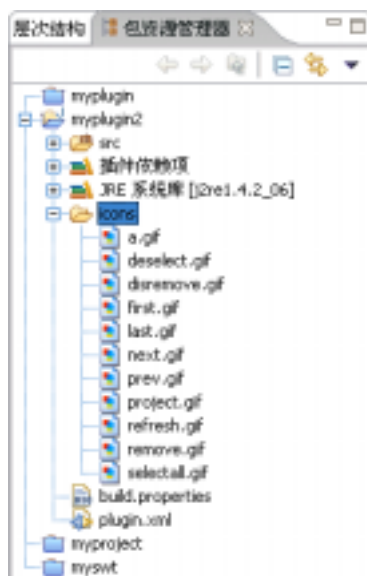


图18.1 图标的路径

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

在上一步的plugin.xml中提前设置了透视图对应的类cn.com.chengang.SamplePerspective，这一步就在包cn.com.chengang中创建此类。透视图类必须实现IPerspectiveFactory接口，此接口只有一个方法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”。在弹出窗口中，可以看到一个名为“myplugin透视图”的项，如下图18.2所示。



图18.2 选择透视图

选择并打开“myplugin透视图”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，下面就会往这个透视图加入两个视图。

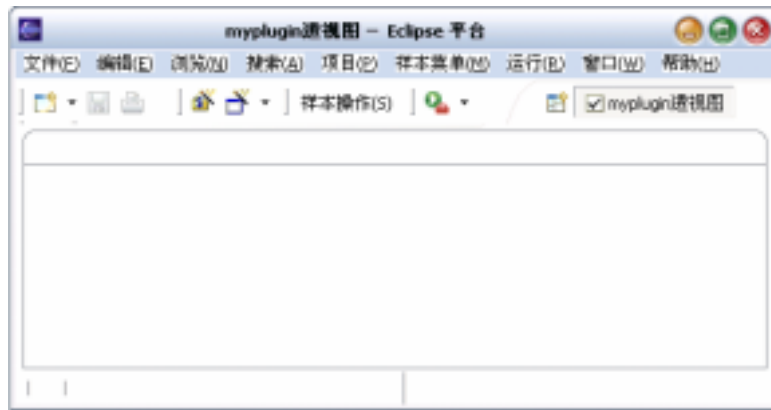


图18.2 myplugin透视图的效果图

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、透视图界面是多么的简单，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)

接着上一节的内容，给透视图加入两个视图，实现的步骤如下：

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>
```

说明：

- org.eclipse.ui.views是视图的扩展点
- <category>...</category> - 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：主菜单“窗口 显示视图 其他”，图18.3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的所有插件中唯一。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 - 表示本视图属于哪个组，与上面<category>项的id值相同
- <view>的class - 视图所对应的类（还没编写，下一步将完成这两个类）
- <view>的id - 视图标识，建议设置成和class一样的名称。



图18.3 显示视图窗口

18.2.2 创建视图类

在上一步的plugin.xml中提前设置了视图对应的类：cn.com.chengang.View1、View2，这一步就来在包cn.com.chengang中创建这两个视图类。

视图的类必须继承抽象类ViewPart，此类有两个方法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。本例将通过修改这个方法，将两个视图加入的透视图。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");
    /*
     * 将上一节所建的action（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18.4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18.4 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。比如，在上一节的界

面中，单击视图1列表中的某项时，视图2的文本框也做相应显示。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，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到此对象的方法。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().getActiveWorkbenchWindow().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。

象"cn.com.chengang.View2"这种标识符在系统开发中会经常性的用到，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，否则把标识符的字符串分散在代码中，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：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18.5所示。同时再为列表添加一个右键菜单，给读者用来和视图的下拉菜单进行比较阅读。

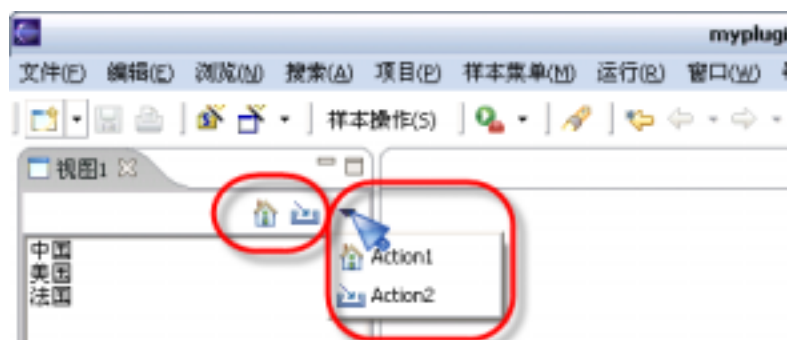


图18.5 效果图

18.4.1 创建ActionGroup类

加入菜单和按钮的方法和SWT/JFace组件的一样。先创建一个ActionGroup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(IWorkbenchGraphicConstants.IMG_ETOOL_HOME
_NAV);

            setHoverImageDescriptor(imageDesc);
            setText("Action1");
        }
        public void run() {}
    }

    private class Action2 extends Action {
        public Action2() {
            ImageDescriptor imageDesc =
WorkbenchImages.getImageDescriptor(IWorkbenchGraphicConstants.IMG_ETOOL_IMPO
RT_WIZ);

            setHoverImageDescriptor(imageDesc);
            setText("Action2");
        }
        public void run() {}
    }
}

```

程序说明：

- 本程序中含有两个Action类：Action1、Action2，和以往的动作不同之处在于它的图像描述符是直接来自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。在下一步时，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中增加了三个方法，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。这种效果就象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18.6所示：

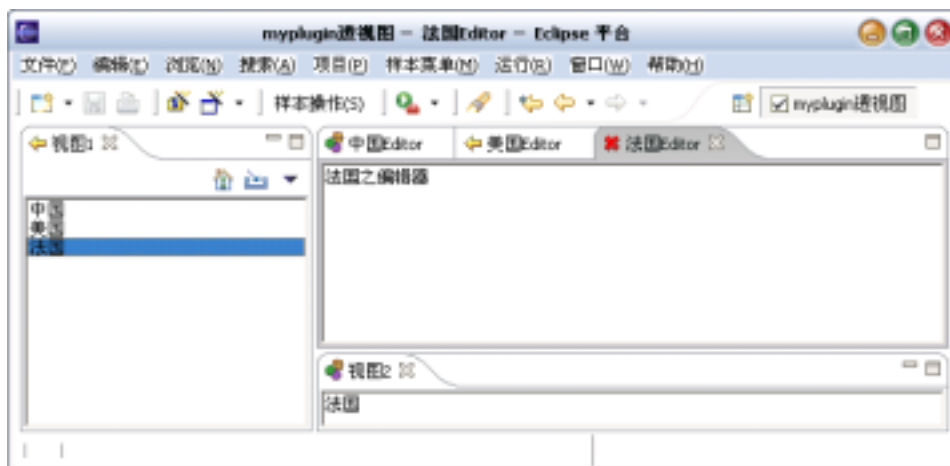


图18.6 编辑器效果图

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

在上一步的plugin.xml中，提前设置了编辑器对应的类cn.com.chengang.ChinaEditor和UsaEditor、FranceEditor，这一步就来在包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个方法，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，则会自动使用一个缺省的图标
        //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，但该方法参数却不是id标识的字串，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，如下图18.7所示。



图18.7 编辑器的组成

在这一步将要创建三个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，请读者查阅随书光盘的代码。

```

//----- 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，不需象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，则代码就会相当的长。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，但以足够应付大部份开发的需要。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

在上一节将ChinaEditor类继承EditorPart抽象类时，只实现了两个方法：init、

createPartControl, 本节将通过“EditorPart方法的执行情况”、“方法的作用及含义”、“一个实例”来逐步讲解其他的五个方法。

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, 我们在类的每一个方法前加上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, 并选择了主菜单“文件 保存”或按钮Ctrl+S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, 比如, 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, 所以不要在这个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()

在这个方法中编写保存编辑器的代码, 当选择主菜单“文件 保存”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此方法。

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”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18.8所示):



图18.8 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个方法中，当按钮Ctrl+S时会的执行此方法。
     * 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.addPropertyChangeListener(new IPropertyChangeListener() {
    //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”打开首选项窗口，如图18.9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。

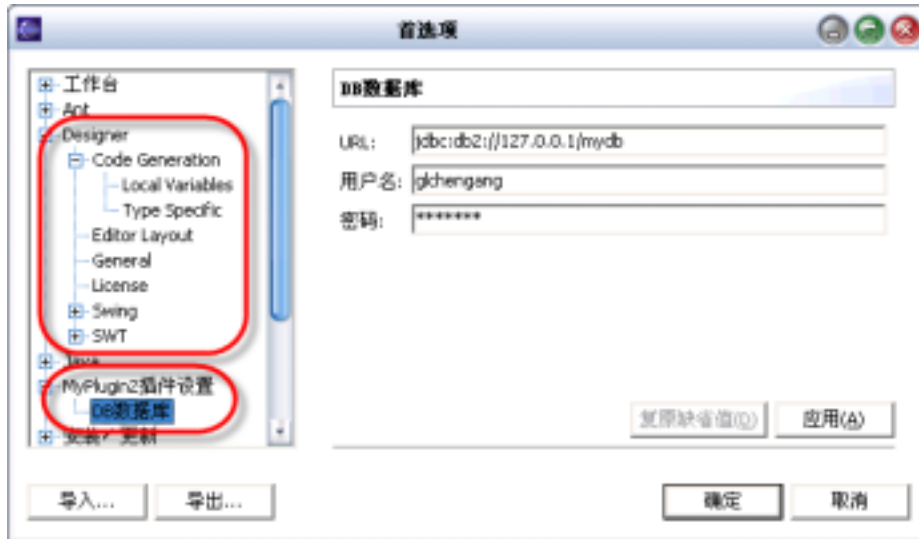


图18.9 首选项

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 - 首选项的树结点的名称
- class - 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 - 首选项的树结点标识，建议设置成和class一样的名称。
- category - 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

在上一步的plugin.xml中提前设置了首选项结点对应的类RootPreferencePage、DBPreferencePage，这一步就来在包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，接口只有一个方法init，抽象类则有一些“首选项”窗口的按钮的执行方法。本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缺省值
    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().getPreferenceStore());
    }

    /**
     * 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();
        /*
         * 从取出以前保存的值,并将其设置到文本框中,如果取出值为空
         * 或者是空字符串,则填入缺省值。
         */
        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“复原缺省值”按钮时将执行此方法, 取出缺省值设置到文本框中。
     */

```

```

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”项，将其中的密码删除后可得到如图18.10的出错效果：



图18.10 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18.11，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。

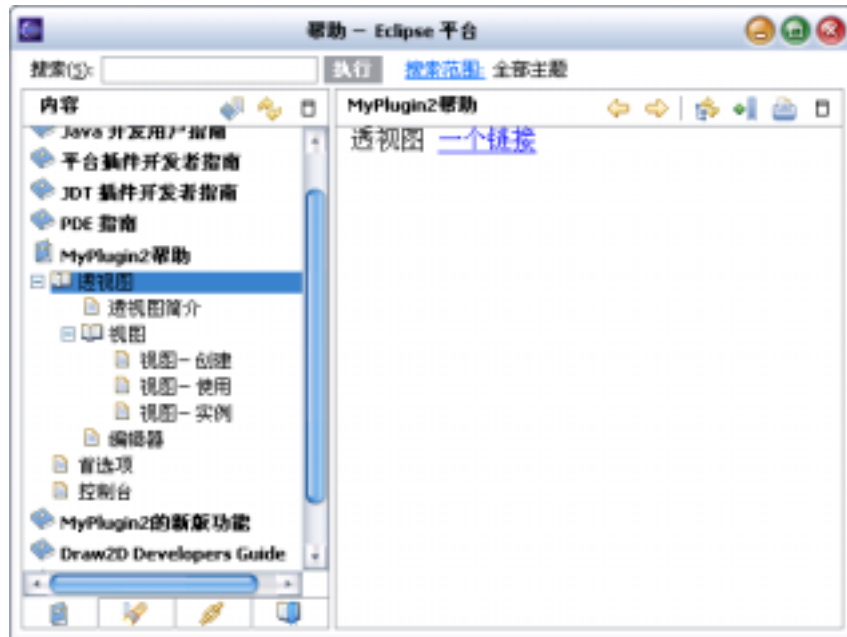


图18.11 帮助

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 - 帮助目录项的设定
- toc的file - 指定帮助目录文件（还没编写，下一步将完成这两xml文件）
- toc的primary - 该项帮助目录是否显示在帮助窗口中，true为显示。
- 如上图18.11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目录同级，如图18.12所示。

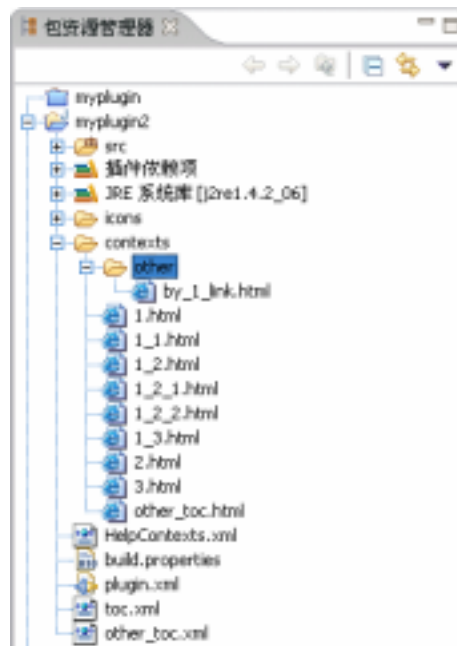


图18.12 目录结构图

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，下一步将完成它们）
- 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，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目录同级，目录结构如上图18.12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，其效果如下图18.13所示。如果组件设置有弹出信息，按帮助键F1就可以激活它。

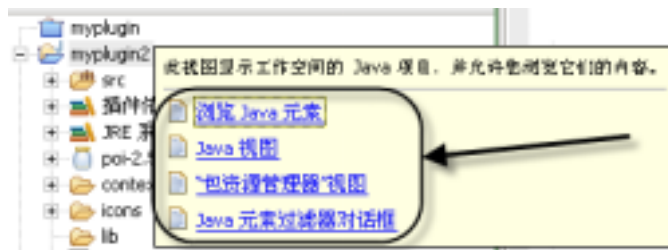


图18.13 弹出信息

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>
```

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，可以和上一节的帮助文件共用，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设置的效果如下图18.14所示



图18.14 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项设置的都是上一节的帮助文件，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，其代码如下：

```
//----- 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18.15所示：



图18.15 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格式的帮助用户也可以共用，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列表：

- 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流程来走。