eclipse插件开发节点说明

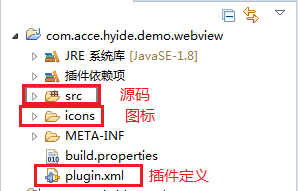
zjl

2018-10-08T00:00:00.000Z

# 项目说明

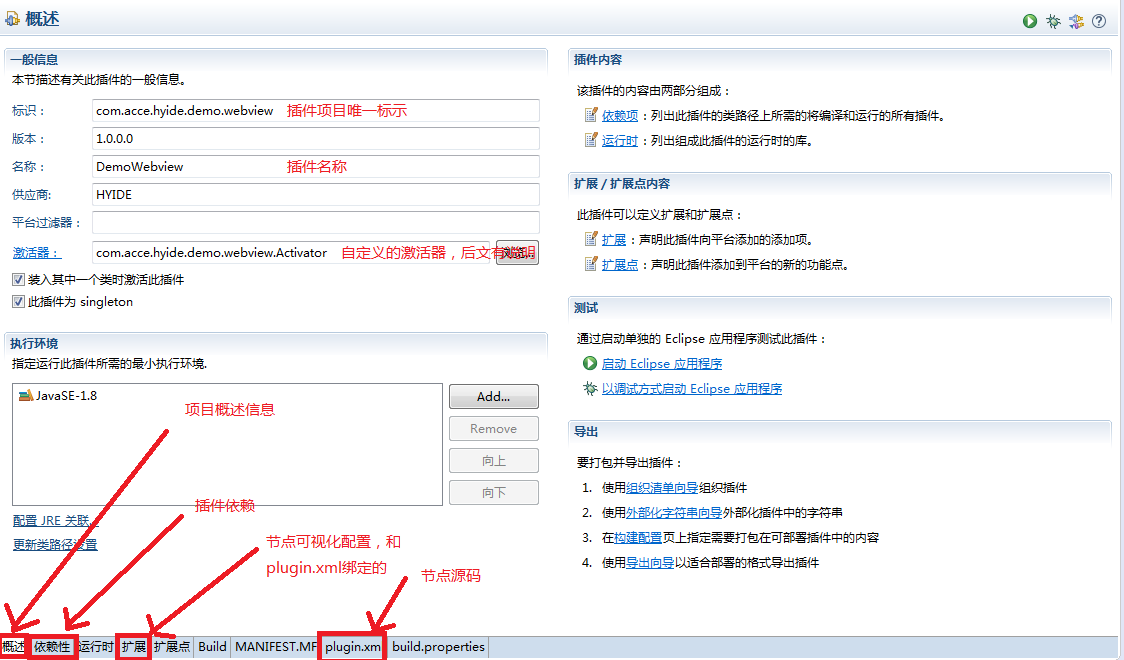
* 插件项目结构说明

插件项目主要有三个文件夹和文件：src、icons、plugin.xml,如下图所示。

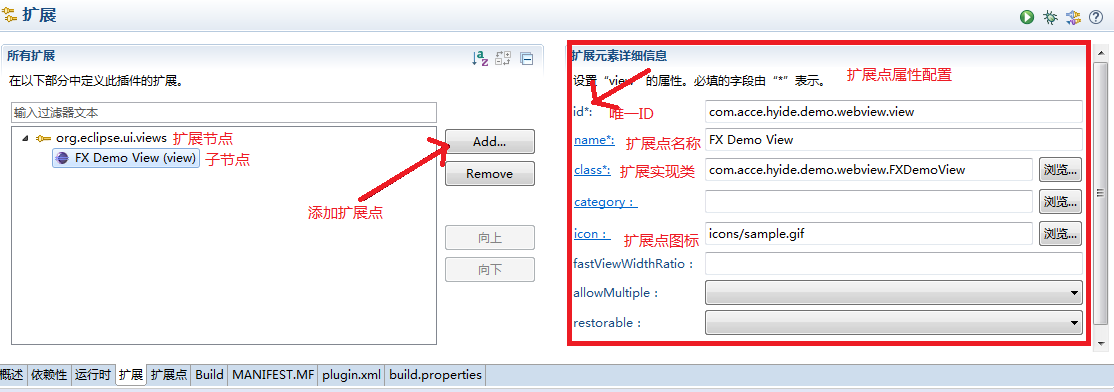


* plugin.xml 说明

Eclipse 插件 都是通过 节点 拓展的，通过节点 id 关联对应的 类 ， 以实现设计的插件功能。所有节点定义都在plugin.xml中，

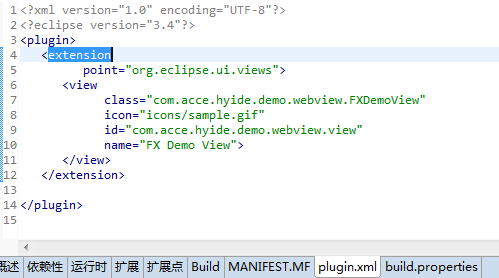


* 添加扩展点



点击Add后，会弹出新建扩展点导航，帮助快速新建扩展点，扩展点属性中，标记 \* 的，都是必填项目，标记 ！的是不推荐填写项目，无标记的是选填项目。

扩展信息在plugin.xml中保存，一般不建议手动修改xml，容易出错。



# IDE启动

## IDE启动相关

通过此部分节点说明,可简单定义你的IDE外观和欢迎页等简单功能。

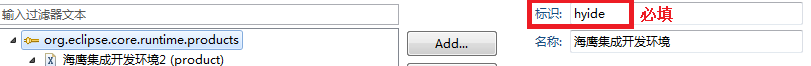
* 节点概览

|  |  |  |
| --- | --- | --- |
| 节点 | 关键节点 | 说明 |
| org.eclipse.core.runtime.products |  | 产品定义 |
| org.eclipse.ui.startup |  | 启动前操作 |
| org.eclipse.ui.themes |  | 产品主题（颜色、字体） |
| org.eclipse.ui.intro |  | 简介 |
| org.eclipse.ui.intro.config |  | 简介配置 |
| org.eclipse.ui.decorators |  | 资源文件图标更新 |

### IDE启动画面和名称 org.eclipse.core.runtime.products

此扩展可自定义启动欢迎界面和标题、名称等。

* 主节点示意图



上图红框中的标识是必填的，系统会根据此标识来查找你的产品定义。

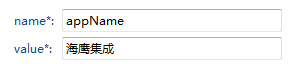
* 节点关系图



* 节点说明

这是一个比较特殊的扩展点，他有且只有一个product节点，并且product的子节点都是通过key-value来和IDE配置意义对应的。如下图

* 参数截图



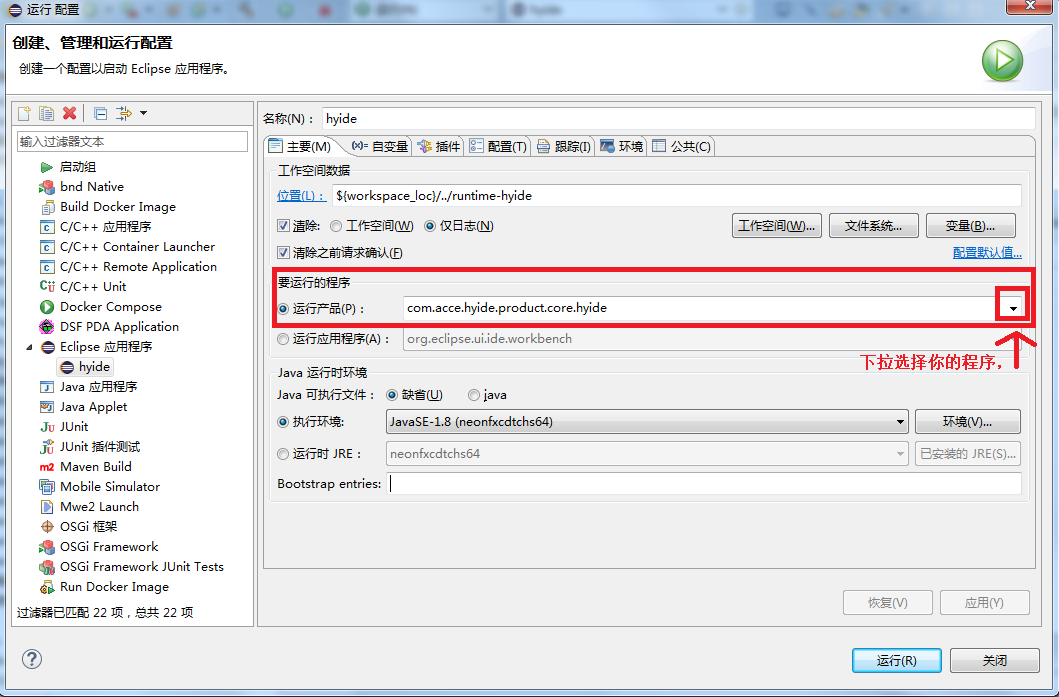
* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| name\* | key值，需要自定义的控件名 |
| value\* | 自定义控件参数 |

例如：appName表示IDE名称，在value填写内容就ok。

* 运行方式

如果想要用自定义产品配置启动Eclipse，需要到 运行->运行配置->要运行的程序中选择你的配置。



选择的程序就是你的插件名称.products的标识

### 启动 org.eclipse.ui.startup

此扩展点用来注册想要在启动时激活的插件。作为 startup 元素的属性给定的类必须实现 org.eclipse.ui.IStartup 接口。一旦启动了工作台，就会从独立的线程中调用 earlyStartup() 方法。如果 startup 元素具有 class 属性，则将把该类实例化并对结果调用 earlyStartup() 方法。

* 节点关系图



* 此扩展点结构比较简单，暂做简要说明

#### startup 启动

ide启动时调用

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| class\* | 启动类，此类需实现 *IStartup* 接口，系统会在独立线程中调用 earlyStartup() 方法。 |

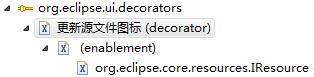
* demo

import org.eclipse.swt.widgets.Display;  
import org.eclipse.ui.IStartup;  
   
public class Startup implements IStartup {  
   
 @Override  
 public void earlyStartup() {  
 Display.getDefault().syncExec(new Runnable() {  
 public void run() {  
 try {  
 System.setProperty("HY-IDE\_Version", "1.0");  
 System.setProperty("HY-PATCH-Version", "1.0.0");  
 } catch (Throwable t) {  
 t.printStackTrace();  
 }  
 }  
 });  
   
 }  
   
}

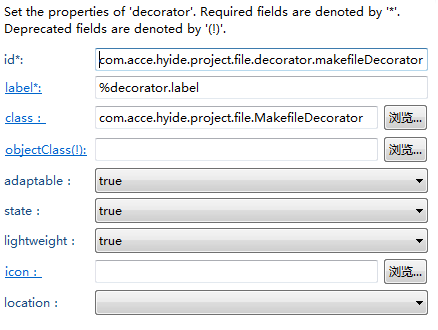
### 资源文件图标更新 org.eclipse.ui.decorators

此扩展点用来将修饰符添加到遵从修饰符管理器的视图。对于 2.1，存在轻量级修饰符的概念，它用来处理修饰符的图像管理。还有可能声明启用时只覆盖图标不需要插件中的实现的轻量级修饰符。 操作的启用和/或可视性分别可使用元素 enablement 和 visibility 定义。这两个元素包含进行求值来确定启用和/或可视性的布尔表达式。

* 节点关系图



其中我们需要了解的是 decorator 子节点参数。 \* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 节点id，建议和class一致 |
| lable\* | 操作名称 |
| objectClass(!) | (**不推荐**)此修饰符将应用于的类的标准名称 |
| class\* | 启动类，此类需实现 *IStartup* 接口，系统会在独立线程中调用 earlyStartup() 方法。 |
| state | 它指示在缺省情况下修饰符是否是打开的 |
| lightweight | 标志指示修饰符是可声明的或者实现 |
| adaptable | 它指示可以适应除其 objectClass 以外的对象的类型是否应该使用此对象添加项,对于轻量级修饰符来说，只要通过适配器管理器定义适应性，就支持对任何 objectClass 的适应性 （请参阅类 org.eclipse.runtime.IAdapterManager）。 |
| icon | 如果修饰符为 lightweight 并且未指定 class，则这是要应用的覆盖图像的路径 |
| location | 如果修饰符为 lightweight，则这是将修饰符应用于的位置。缺省值为 BOTTOM\_RIGHT |

一般情况下，后几个参数不用设置，如图所示即可。

* demo

import org.eclipse.core.resources.IFile;  
import org.eclipse.core.resources.IFolder;  
import org.eclipse.core.resources.IResource;  
import org.eclipse.core.runtime.Platform;  
import org.eclipse.jface.preference.JFacePreferences;  
import org.eclipse.jface.resource.ImageDescriptor;  
import org.eclipse.jface.resource.JFaceResources;  
import org.eclipse.jface.viewers.IDecoration;  
import org.eclipse.jface.viewers.ILabelProviderListener;  
import org.eclipse.jface.viewers.ILightweightLabelDecorator;  
   
import com.acce.hyide.project.file.Activator;  
   
   
public class MakefileDecorator implements ILightweightLabelDecorator {  
 public static final String MAKEFILE\_REMOVED = "icons/informationcenter.png"; //$NON-NLS-1$  
 public static ImageDescriptor MAKEFILE\_REMOVED\_IMAGE;  
 // 初始化ImageDescriptor  
 static {  
 MAKEFILE\_REMOVED\_IMAGE = ImageDescriptor  
 .createFromURL(Platform.getBundle(Activator.PLUGIN\_ID).getEntry(MAKEFILE\_REMOVED));  
 }  
   
 @Override  
 public void addListener(ILabelProviderListener listener) {  
 // TODO 自动生成的方法存根  
   
 }  
   
 @Override  
 public void dispose() {  
 // TODO 自动生成的方法存根  
   
 }  
   
 @Override  
 public boolean isLabelProperty(Object element, String property) {  
 // TODO 自动生成的方法存根  
 return false;  
 }  
   
 @Override  
 public void removeListener(ILabelProviderListener listener) {  
 // TODO 自动生成的方法存根  
   
 }  
   
 @Override  
 public void decorate(Object element, IDecoration decoration) {  
 IResource resource;  
 String projectLocation; // 项目的绝对路径  
 String elementPath; // 被点击文件的相对路径  
   
 if (element instanceof IFile || element instanceof IFolder) {  
 resource = (IResource) element;  
 projectLocation = resource.getProject().getLocation().toString();  
 elementPath = absolutePathToRelativePath(resource.getLocation().toString(),  
 projectLocation);  
 } else {  
 return;  
 }  
   
 // 文件（文件夹左下角）显示x图标；  
 decoration.addOverlay(MAKEFILE\_REMOVED\_IMAGE);  
 // 字体变灰色  
 decoration.setForegroundColor(JFaceResources.getColorRegistry()  
 .get(JFacePreferences.QUALIFIER\_COLOR));  
   
 }  
   
 /\*  
 \* 将绝对路径转为相对路径，不保留第一个"/"，通过调用subString（）方法，去掉第一个“/”  
 \* example  
 \* absolutePath="E:/sylixos/plugin\_workspace/runtime-EclipseApplication/app\_demo/functions/f1.c";  
 \* projectLocation="E:/sylixos/plugin\_workspace/runtime-EclipseApplication/app\_demo";  
 \* return "functions/f1.c";  
 \*/  
 public static String absolutePathToRelativePath(String absolutePath, String projectLocation) {  
 String result = absoluteToRelative(absolutePath, projectLocation);  
 if (result.equals("")) {  
 return "";  
 } else {  
 return result.substring(1);  
 }  
 }  
   
 /\*  
 \* 将绝对路径转为相对路径，保留第一位的“/”  
 \*/  
 public static String absoluteToRelative(String file, String parentFile) {  
 return absoluteToRelative(new File(file), new File(parentFile));  
 }  
   
 /\*  
 \* 将绝对路径转为相对路径，保留第一位的“/”  
 \*/  
 public static String absoluteToRelative(File file, File parentFile) {  
 if (file.equals(parentFile)) {  
 return ""; //$NON-NLS-1$  
 } else {  
 return absoluteToRelative(file.getParentFile(), parentFile) + "/" + file.getName();  
 }  
 }  
   
}

### 产品简介（欢迎页） org.eclipse.ui.intro

此扩展点用来注册负责向新用户介绍产品的特殊工作台部分（称为简介部分）的实现。简介部分通常会在第一次启动产品时显示。还通过此扩展点提供了将简介部分实现与特定产品相关联的规则 因近来前端搭UI是趋势，故此处以配置web页做Demo。

* ui样例：



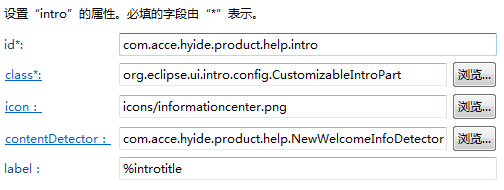
* 节点关系图



#### (简介声明) intro

指定简介。简介是特定于产品的表示，会在产品启动时显示给第一次使用的用户

参数截图：



|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 节点id |
| class\* | 系统类 org.eclipse.ui.intro.config.CustomizableIntroPart |
| icon | 图标路径 |
| contentDetector | 欢迎页检查者，一个继承IntroContentDetector的实现类，可以不写 |
| 说明标题 | 一般为 “欢迎使用” |

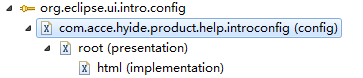
#### (简介与产品绑定) introProductBinding

指定产品与简介之间的绑定。这些绑定确定哪个简介适用于当前产品

|  |  |
| --- | --- |
| 参数 | 说明 |
| productId\* | 绑定的产品id |
| introId\* | 绑定的简介ID |

### 简介配置（欢迎页实现方式） org.eclipse.ui.intro.config

节点关系图



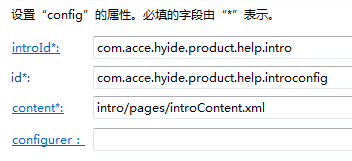
通过config配置欢迎页实现方式

* 节点说明

#### config

指定欢迎页实现方式

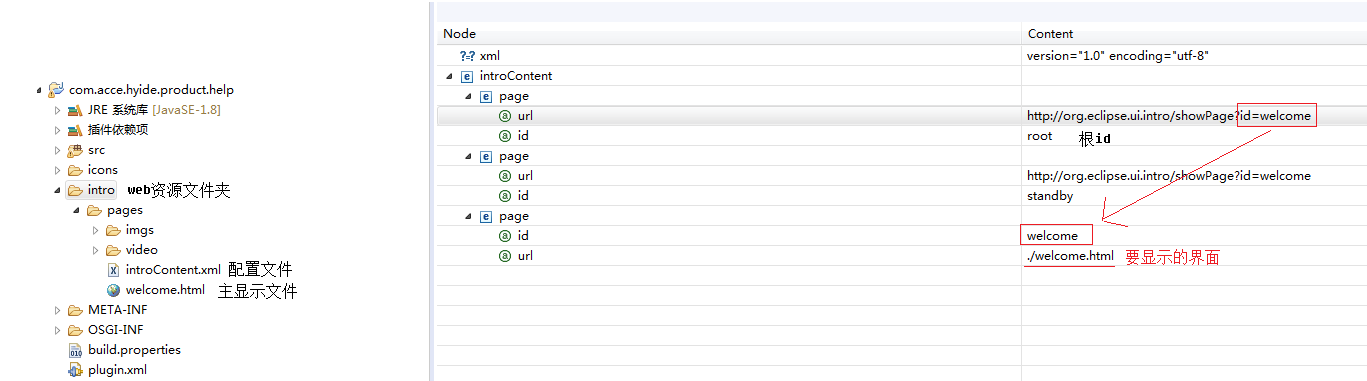
* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| introId\* | 配置的简介ID |
| id\* | 此节点ID |
| content\* | 欢迎页内容，如果是html实现，则填写web配置文件路径 |
| configurer | 标准类的名称，此类支持简介配置的动态方面。此类可以提供替换变量的值和动态组的子代，并可以解析不完整的目标路径。此类必须扩展 org.eclipse.ui.intro.IntroConfigurer。此处未使用，不做说明 |

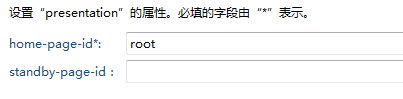
* introcontent.xml 样例：



#### presentation

定义简介部分的表示的所有可能实现的表示元素。它可能定义了一个或多个实现。根据实现的 os/ws 属性，启动时只会有一个实现被选中。否则，没有定义 os/ws 属性的第一项将会被选中。

* 参数截图



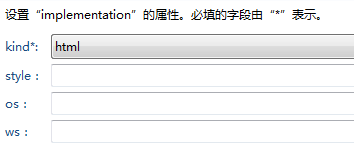
* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| home-page-id | 主（根）页的标识，主页即简介的第一页。此页面可用作构成简介的其他主页面的入口点。 |
| standby-page-id | 用来定义备用页面的标识的可选属性。备用页面将在简介设置为备用时显示给用户。 |

#### implementation

有两个实现。其中一项是基于 SWT 浏览器的，而另一项是基于 UI 表单的。可配置可定制简介部分以便根据当前操作系统和窗口系统选择这两个表示的其中一个。实现的类型可以是 SWT 或 HTML。

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| kind | 指定此实现的类型。SWT 种类指示基于用户 UI 表单的实现，而 HTML 种类指示基于 SWT 浏览器的实现 |
| style | 将应用于此简介表示实现表示的所有页面的共享样式。 |
| os | 在选择表示的实现时使用的可选操作系统规范。它可以是 Eclipse 定义的任何操作系统标识，例如，win32 和 linux 等等（请参阅 org.eclipse.core.runtime.Platform 的 Javadoc）。 |
| ws | 在选择表示的实现时使用的可选窗口系统规范。它可以是 Eclipse 定义的任何窗口系统标识 |

# 系统菜单、工具条和弹出菜单

## 主菜单、工具条、弹出菜单和首选项

这里介绍几个实现常用菜单的相关节点实现方法。值得注意的是：这里的节点不会通过class参数直接绑定实现功能的类，而是大多通过 **org.eclipse.ui.commands** 节点的子节点**command**来链接实现的类。所以这里也附带解释org.eclipse.ui.commands节点。

* 节点概览

|  |  |  |
| --- | --- | --- |
| 节点 | 关键节点 | 说明 |
| org.eclipse.ui.menus | menuContribution、menu、toolbar、command | 实现菜单栏和工具条 |
| org.eclipse.ui.popupMenus | objectContribution、menu、separator、action | 右键资源弹出菜单 |
| org.eclipse.ui.propertyPages | page | 右键资源属性页 |
| org.eclipse.ui.preferencePages | page | 首选项 |
| org.eclipse.core.runtime.preferences | initializer | 首选项初始化 |
| org.eclipse.ui.commands | command | 操作指令 |
| org.eclipse.ui.handers | hander | 指令操作者 |
| org.eclipse.ui.actionSets | actionSets | 操作集 |

### 菜单栏、工具条、右键任意弹出菜单 org.eclipse.ui.menus

menus节点可配置菜单和工具栏项，此节点有丰富的配置选项，有多种实现方式，详见各 **参数说明**

* 参数说明

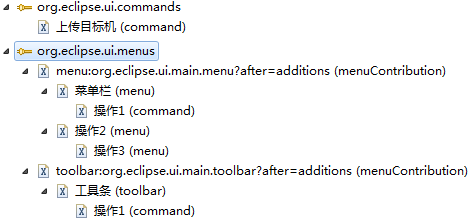
|  |  |
| --- | --- |
| 关键节点 | 说明 |
| menuContribution | 控制菜单位置，是菜单栏还是在工具条 |
| menu | 菜单栏控件 |
| toolbar | 工具条控件 |
| command | 控件链接的具体实现类 |

* 视图说明



#### 节点关系和继承

* 关系图



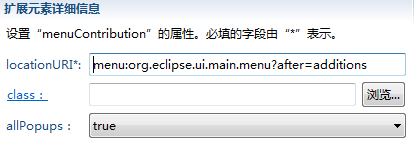
* 节点概要

|  |  |
| --- | --- |
| 节点 | 说明 |
| menuContribution | 可设置子节点是菜单栏或工具条 |
| menu | 菜单栏节点，可多层**嵌套** |
| toolbar | 工具条节点，实现类和 *command* 一样。 |
| command | *menu* 的执行链接，通过此链接class，实现具体菜单操作，当然，也可以通过 *menu* 内的 属性直接设置。 |

#### 节点解析

##### 子节点:menuContribution

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| locationURI\* | 表示菜单位置 **menu:org.eclipse.ui.main.menu?after=additions** 表示此节点为菜单栏，after的值是表示菜单的位置；**toolbar:org.eclipse.ui.main.toolbar?after=additions** 表示此节点为工具条；**popup:org.eclipse.ui.popup.any?after=additions**表示任意右键弹出菜单；**menu:help?after=additions**表示菜单栏的帮助下。具体信息见视图说明 |
| class | 这里的class不是必要参数，一般不设置 |

##### 子节点: menu

* 参数截图



* 参数说明

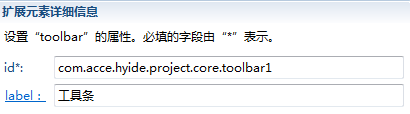
|  |  |
| --- | --- |
| 参数 | 说明 |
| lable\* | 这里的lable就是显示在**菜单栏/右键弹出菜单**的具体菜单名，用户通过此进行交互 |
| id | com.xxx.menu // 非必须 |
| commandId | 默认command 链接，非必须 |

* **小技巧** 快速被你的菜单添加快捷键，如Alt+G

只需在 menu 的lable 属性中，添加(&G),如，参数截图所示

##### 子节点: toolbar

* 参数截图

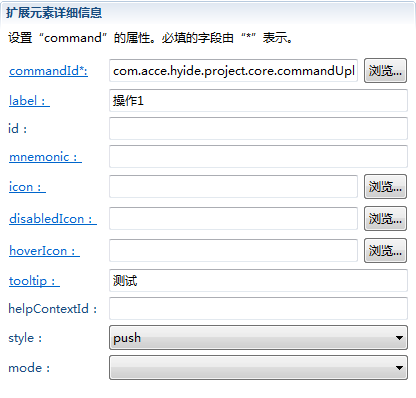


* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| lable | 这里的lable就是显示在**工具栏**的具体菜单名，用户通过此进行交互 |
| id\* | 必须id |

##### 子节点: command

* 参数截图



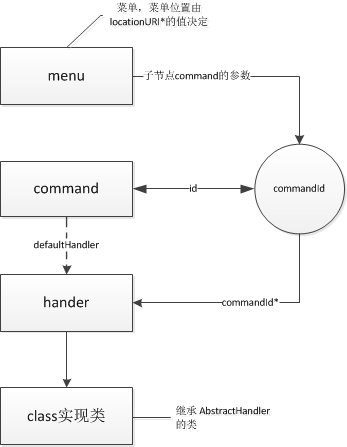
* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| commandId\* | 这里的lable就是显示在工具栏的具体菜单名，用户通过此进行交互 |
| lable | 标识此操作的名称，显示在菜单栏的操作项上 |
| id | 非必须的 |
| tooltip | 鼠标停留在控件上时的提示文本 |

* 节点说明

这里的 command(记为 A) 和 节点 **org.eclipse.ui.commands** 中的 command(记为B) 作用是不一样的，A 通过 参数 [commandId] 链接 B，B会通过 参数 [defaultHandler] 链接对应的[hander]来指向具体 calss。

* 菜单的command的实现方式



### 链接指令 org.eclipse.ui.commands

commands节点是一个连接菜单节点和实现类的纽带，此节点是一个指令，每个指令绑定一个实现方法，其他节点通过调用此节点，实现指定的功能。详见 **参数说明**

* 节点关系图

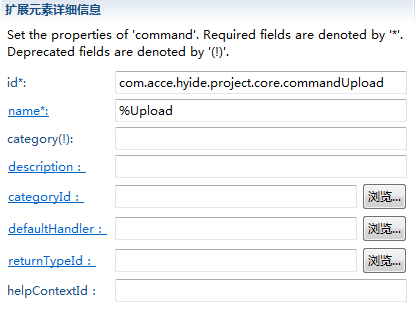


* 参数说明

|  |  |
| --- | --- |
| 关键节点 | 说明 |
| category | 一个分类菜单，可以不设 ，暂时未涉及 |
| command | 一个具体的操作指令，在defaultHandler中设置指令执行对象hander |

#### 子节点 command

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| name\* | 操作指令名称 |
| id\* | id 名称 ，如果没有设置 defaultHandler ，则 *handler* 会通过此id进行关联 |
| defaultHandler | 默认hander的 id ，可以不设置 |

### 指令处理 org.eclipse.ui.handlers

handlers是commands的是实现节点，commands会通过此节点绑定类。详见 **节点说明**

#### 子节点 handle

* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| commandId\* | command的索引id |
| class\* | 绑定的操作实现类,此类需要继承 **AbstractHandler**。会自动调用 实现类的 execute 函数 |

##### handle - class 实现类

public class DebugHandler extends AbstractHandler {  
   
 @Override  
 public Object execute(ExecutionEvent event) throws ExecutionException {  
 // your du  
 return null;  
 }  
   
}

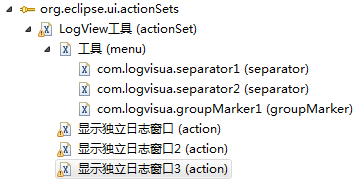
### 操作集 org.eclipse.ui.actionSets

此节点通过 子节点的参数*definitionId* 链接*command*，为其他节点如menu、tollbar等 添加 实现方式。

* 关键节点

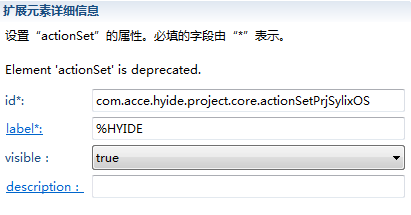
|  |  |
| --- | --- |
| 关键节点 | 说明 |
| actionSet | 类似 menu 的作用 |
| action | 右键资源弹出菜单 |

* 节点关系图



#### actionSet

* 参数截图

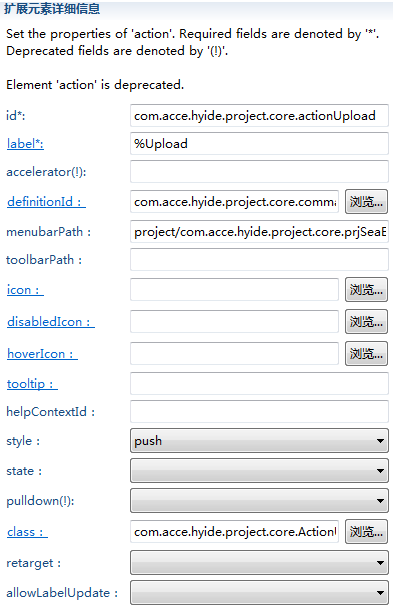


* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 节点id |
| lable | 类似于 menu 的名称 |

#### action

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 节点id |
| lable | 操作项名称 |
| definitionId | org.eclipse.ui.command的子节点 command的id 通过command 链接操作实现类 |
| class | 操作实现类。且通过此节点实现的 类 ，不需要做其他继承关系 |
| menubarPath | 操作项显示位置，一般不直接设置，会被command等其他位置覆盖 |

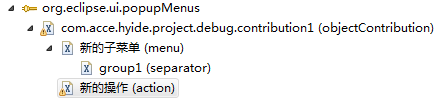
### 右键资源弹出菜单 org.eclipse.ui.popupMenus

popupMenus可实现定制右键资源弹出菜单。详见 **参数说明**

* 节点概要

|  |  |
| --- | --- |
| 关键节点 | 节点说明 |
| objectContribution | 菜单弹出条件 |
| menu | 右键菜单项，属于 objectContribution |
| separator | 菜单分组，属于menu |
| action | 菜单操作项 ，属于objectContribution |

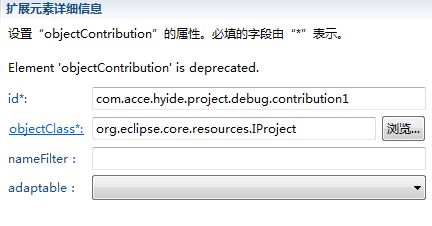
* 节点关系图



#### 节点解析

##### 菜单弹出条件 objectContribution

* 参数截图



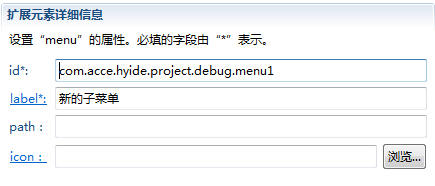
* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | id |
| objectClass\* | 菜单弹出条件：org.eclipse.core.resources.IFile选中文件时弹出；org.eclipse.core.resources.IProject项目弹出；其他参数类同 |

##### 菜单 menu

objectContribution -> menu

* 参数截图



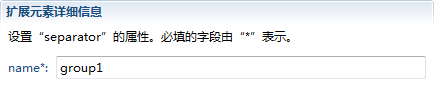
* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | id |
| lable\* | 菜单名称 |

##### 菜单分组 separator

menu ->separator

* 参数截图



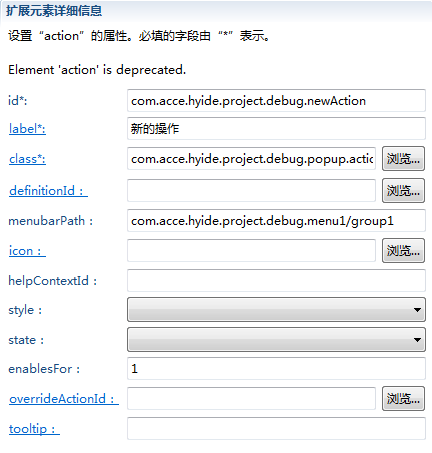
* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| name\* | 菜单分组，group |

##### 菜单操作项 action

objectContribution -> action

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | id |
| lable\* | 操作项名称 |
| class\* | 实现类 |
| menubarPath | 操作项的菜单位置。如：menuId/separatorId |

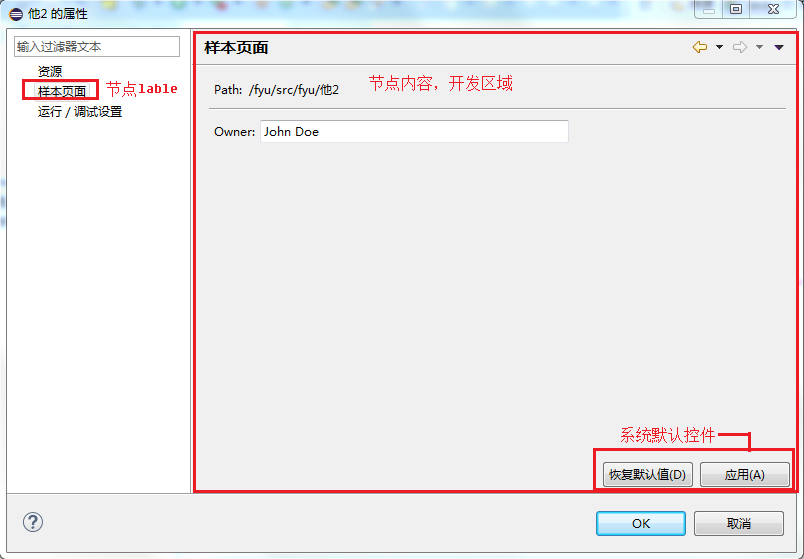
### 右键任意弹出菜单 org.eclipse.ui.menus

同样在[**org.eclipse.ui.menus**]节点下，由子节点[*menuContribution*]属性控制。详细说明见 上文 **org.eclipse.ui.menus**节点说明。

### 资源右键属性页 org.eclipse.ui.propertyPages

|  |  |
| --- | --- |
| 关键节点 | 说明 |
| page | 提供一个页面，用于展示或设置项目或文件属性 |

* 视图说明



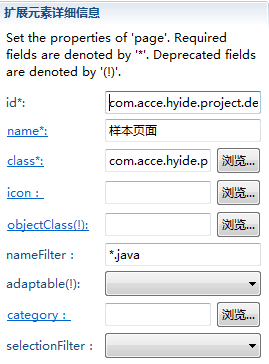
#### 节点关系



#### 节点解析

##### page

* 主要参数截图



* 参数说明

|  |  |
| --- | --- |
| 关键参数 | 说明 |
| id\* | 建议为class名 |
| name\* | 页面名称lable |
| class\* | 实现类名，实现page的类要继承PropertyPage,系统会自动传递 Composite parent,以提供UI布局，Demo见下文 |
| nameFilter | 资源过滤器 |
| category | 页面分类，分类category的Id |

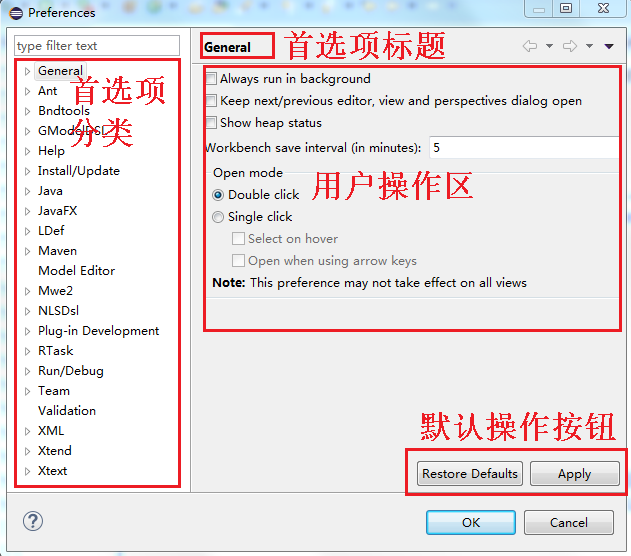
###### page - class 实现类

import org.eclipse.core.resources.IResource;  
import org.eclipse.core.runtime.CoreException;  
import org.eclipse.core.runtime.QualifiedName;  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.layout.GridData;  
import org.eclipse.swt.layout.GridLayout;  
import org.eclipse.swt.widgets.Composite;  
import org.eclipse.swt.widgets.Control;  
import org.eclipse.swt.widgets.Label;  
import org.eclipse.swt.widgets.Text;  
import org.eclipse.ui.dialogs.PropertyPage;  
   
public class SamplePropertyPage extends PropertyPage {  
   
 private static final String PATH\_TITLE = "Path:";  
 private static final String OWNER\_TITLE = "&Owner:";  
 private static final String OWNER\_PROPERTY = "OWNER";  
 private static final String DEFAULT\_OWNER = "John Doe";  
   
 private static final int TEXT\_FIELD\_WIDTH = 50;  
   
 private Text ownerText;  
   
 /\*\*  
 \* Constructor for SamplePropertyPage.  
 \*/  
 public SamplePropertyPage() {  
 super();  
 }  
   
 /\* 交互UI 和数据初始化 \*/  
 /\*\*  
 \* @see PreferencePage#createContents(Composite)  
 \*/  
 protected Control createContents(Composite parent) {  
 Composite composite = new Composite(parent, SWT.NONE);  
 GridLayout layout = new GridLayout();  
 composite.setLayout(layout);  
 GridData data = new GridData(GridData.FILL);  
 data.grabExcessHorizontalSpace = true;  
 composite.setLayoutData(data);  
   
 addFirstSection(composite);  
 addSeparator(composite);  
 addSecondSection(composite);  
 return composite;  
 }  
   
   
   
 /\* 恢复默认值 按钮 响应 \*/  
 protected void performDefaults() {  
 super.performDefaults();  
 // Populate the owner text field with the default value  
 ownerText.setText(DEFAULT\_OWNER);  
 }  
   
 /\* 应用 按钮 响应 \*/  
 public boolean performOk() {  
 // store the value in the owner text field  
 try {  
 ((IResource) getElement()).setPersistentProperty(  
 new QualifiedName("", OWNER\_PROPERTY),  
 ownerText.getText());  
 } catch (CoreException e) {  
 return false;  
 }  
 return true;  
 }  
   
 private Composite createDefaultComposite(Composite parent) {  
 Composite composite = new Composite(parent, SWT.NULL);  
 GridLayout layout = new GridLayout();  
 layout.numColumns = 2;  
 composite.setLayout(layout);  
   
 GridData data = new GridData();  
 data.verticalAlignment = GridData.FILL;  
 data.horizontalAlignment = GridData.FILL;  
 composite.setLayoutData(data);  
   
 return composite;  
 }  
   
 private void addFirstSection(Composite parent) {  
 Composite composite = createDefaultComposite(parent);  
   
 //Label for path field  
 Label pathLabel = new Label(composite, SWT.NONE);  
 pathLabel.setText(PATH\_TITLE);  
   
 // Path text field  
 Text pathValueText = new Text(composite, SWT.WRAP | SWT.READ\_ONLY);  
 pathValueText.setText(((IResource) getElement()).getFullPath().toString());  
 }  
   
 private void addSeparator(Composite parent) {  
 Label separator = new Label(parent, SWT.SEPARATOR | SWT.HORIZONTAL);  
 GridData gridData = new GridData();  
 gridData.horizontalAlignment = GridData.FILL;  
 gridData.grabExcessHorizontalSpace = true;  
 separator.setLayoutData(gridData);  
 }  
   
 private void addSecondSection(Composite parent) {  
 Composite composite = createDefaultComposite(parent);  
   
 // Label for owner field  
 Label ownerLabel = new Label(composite, SWT.NONE);  
 ownerLabel.setText(OWNER\_TITLE);  
   
 // Owner text field  
 ownerText = new Text(composite, SWT.SINGLE | SWT.BORDER);  
 GridData gd = new GridData();  
 gd.widthHint = convertWidthInCharsToPixels(TEXT\_FIELD\_WIDTH);  
 ownerText.setLayoutData(gd);  
   
 // Populate owner text field  
 try {  
 String owner =  
 ((IResource) getElement()).getPersistentProperty(  
 new QualifiedName("", OWNER\_PROPERTY));  
 ownerText.setText((owner != null) ? owner : DEFAULT\_OWNER);  
 } catch (CoreException e) {  
 ownerText.setText(DEFAULT\_OWNER);  
 }  
 }  
   
   
}

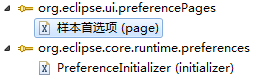
### 首选项 org.eclipse.ui.preferencePages

此节点可通过模板创建:Add->Extension Wizards->Preference Page(首选项) 创建成功后，会有以下节点，对应的class也会自动创建。下面给出节点参数简要说明。

* UI实例



* 节点关系

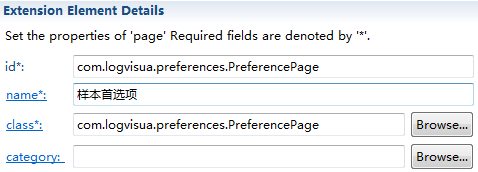


* 节点说明

|  |  |  |
| --- | --- | --- |
| 节点 | 关键节点 | 说明 |
| org.eclipse.ui.preferencePages | page | 首选项 |
| org.eclipse.core.runtime.preferences | initializer | 首选项初始化 |

#### 首选项UI org.eclipse.ui.preferencePages

* 参数截图



* 参数说明

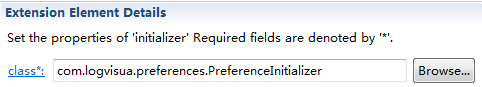
|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | id，建议和实现类名一样 |
| name\* | 页面lab名字 |
| class\* | UI实现类,此类要继承 FieldEditorPreferencePage 并 implements IWorkbenchPreferencePage |
| category | 分类节点id |

* demo：

// 组件key值，首选项数据以key-Value保存，所以key值不能一样  
public class PreferenceConstants {  
   
 public static final String P\_PATH = "pathPreference";  
   
 public static final String P\_BOOLEAN = "booleanPreference";  
   
 public static final String P\_CHOICE = "choicePreference";  
   
 public static final String P\_STRING = "stringPreference";  
   
}  
   
// ui实现  
public class PreferencePage  
 extends FieldEditorPreferencePage  
 implements IWorkbenchPreferencePage {  
   
 public PreferencePage() {  
 super(GRID);  
   
 setPreferenceStore(Activator.getDefault().getPreferenceStore());  
 setDescription("A demonstration of a preference page implementation");  
 }  
   
 public void createFieldEditors() {  
 addField(new DirectoryFieldEditor(PreferenceConstants.P\_PATH,  
 "&Directory preference:", getFieldEditorParent()));  
 addField(  
 new BooleanFieldEditor(  
 PreferenceConstants.P\_BOOLEAN,  
 "&An example of a boolean preference",  
 getFieldEditorParent()));  
   
 addField(new RadioGroupFieldEditor(  
 PreferenceConstants.P\_CHOICE,  
 "An example of a multiple-choice preference",  
 1,  
 new String[][] { { "&Choice 1", "choice1" }, {  
 "C&hoice 2", "choice2" }  
 }, getFieldEditorParent()));  
 addField(  
 new StringFieldEditor(PreferenceConstants.P\_STRING, "A &text preference:", getFieldEditorParent()));  
 }  
   
   
 public void init(IWorkbench workbench) {  
 }  
   
}

#### 首选项初始化 org.eclipse.core.runtime.preferences

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| class\* | UI实现类，需继承 AbstractPreferenceInitializer |

* demo： :::{custom-style=“代码”}

public class PreferenceInitializer extends AbstractPreferenceInitializer {  
   
 public void initializeDefaultPreferences() {  
 IPreferenceStore store = Activator.getDefault().getPreferenceStore();  
   
 // 设置默认值  
 store.setDefault(PreferenceConstants.P\_BOOLEAN, true);  
 store.setDefault(PreferenceConstants.P\_CHOICE, "choice2");  
 store.setDefault(PreferenceConstants.P\_STRING,  
 "Default value");  
   
 // key通过store.getInt(Key);等来获取相应的值，还有getString()等。  
 }  
}

:::

#### Activator配置

前面的2个节点的配置是首选项的关节节点，但我们依然需要实现一个Activator，否则会在

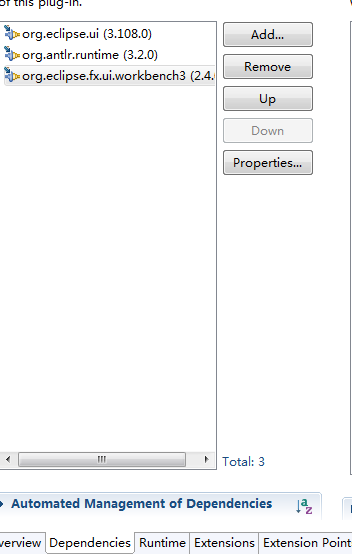
setPreferenceStore(Activator.getDefault().getPreferenceStore());

时，报空指针异常，并无法显示界面。 下面是Activator配置。

* 建立 Activator.class

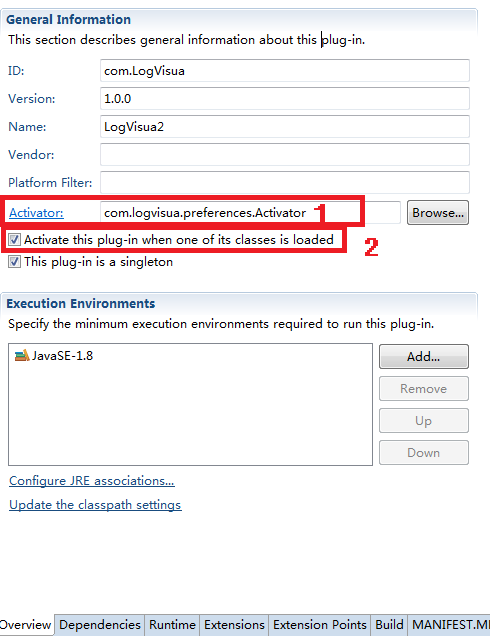
import org.eclipse.ui.plugin.AbstractUIPlugin;  
import org.osgi.framework.BundleContext;  
   
/\*\*  
 \* The activator class controls the plug-in life cycle  
 \*/  
public class Activator extends AbstractUIPlugin {  
   
 // The plug-in ID  
 public static final String PLUGIN\_ID = "com.LogVisua"; //$NON-NLS-1$  
   
 // The shared instance  
 private static Activator plugin;  
   
 /\*\*  
 \* The constructor  
 \*/  
 public Activator() {  
 }  
   
 /\*  
 \* (non-Javadoc)  
 \* @see org.eclipse.ui.plugin.AbstractUIPlugin#start(org.osgi.framework.BundleContext)  
 \*/  
 public void start(BundleContext context) throws Exception {  
 super.start(context);  
 plugin = this;  
 }  
   
 /\*  
 \* (non-Javadoc)  
 \* @see org.eclipse.ui.plugin.AbstractUIPlugin#stop(org.osgi.framework.BundleContext)  
 \*/  
 public void stop(BundleContext context) throws Exception {  
 plugin = null;  
 super.stop(context);  
 }  
   
 /\*\*  
 \* Returns the shared instance  
 \*  
 \* @return the shared instance  
 \*/  
 public static Activator getDefault() {  
 return plugin;  
 }  
   
}

* 添加依赖 org.eclipse.fx.ui.workbench3



* 配置xml

在plugin.xml的Overvie页中，配置 **Activator** 并勾选 *Activate this plug-in when one of its class is loaded*使插件启动时装载Activator

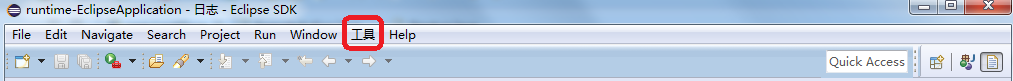


在1处Borwes新建的Activator.class,并勾选2.

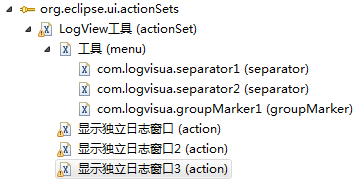
### 操作集 org.eclipse.ui.actionSets 更方便的添加菜单、菜单项和工具栏按钮添

此扩展点用来将菜单、菜单项和工具栏按钮添加到“工作台”窗口中的公共区域。将这些添加项统称为操作集并由定制透视图的用户安排出现在“工作台”窗口中。入下图所示

* UI实例



* 节点关系图

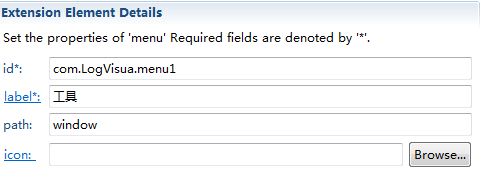


#### 节点解析

此节点只有1个二级节点：**actionSet**。actionSet有2个子节点，分别为menu菜单和action操作。action通过menubarPath和toolbarPath确定位置。和分类

##### actionSets.menu

* 参数截图



* 参数说明

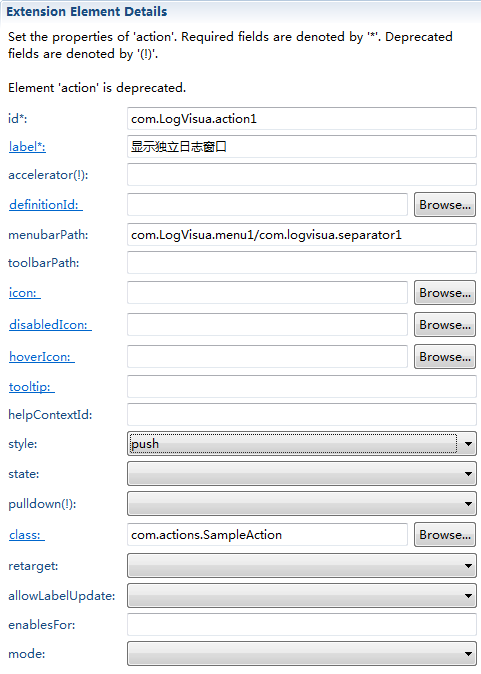
|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | id，建议和实现类名一样 |
| label\* | 菜单名称 |
| path\* | 菜单显示位置，**window** 表示菜单位于菜单栏 |

menu还有子节点separator和groupMarker，这2都是菜单分类，不同分类下的actionSet会通过 横线 分隔开，menu只有name一个属性，这里就不做说明了。

##### actionSets.action

操作项，连接着一个实现类

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | id，建议和实现类名一样 |
| label\* | 操作名称 |
| menubarPath | 显示在菜单栏的位置，与 toolbarPath 互不干扰 |
| toolbarPath | 显示在工具条的位置，与 menubarPath 互不干扰 |
| class | 需要实现 IWorkbenchWindowActionDelegate 接口，在public void run(IAction action)中开始操作。IAction是调用操作的按钮 |

demo:

public class SampleAction implements IWorkbenchWindowActionDelegate{  
   
 @Override  
 public void run(IAction action) {  
 String strActionName = action.getText();  
 System.out.println(strActionName );//按钮名称  
 System.out.println(action.getId() );// 按钮id  
 }  
   
 @Override  
 public void selectionChanged(IAction action, ISelection selection) {  
 // TODO Auto-generated method stub  
   
 }  
   
 @Override  
 public void dispose() {  
 // TODO Auto-generated method stub  
   
 }  
   
 @Override  
 public void init(IWorkbenchWindow window) {  
 // TODO Auto-generated method stub  
   
 }  
   
}

# 工作台视图

## 工作台视图(透视图)相关节点说明

工作台是用户使用最频繁的部件，如代码编辑视图、调试视图都在工作台上。通过此部分说明，可实现定制简单工作台视图。主要步骤有：视图扩展和注册、工作台透视图位置布局，透视图功能实现。详见个子结点说明。

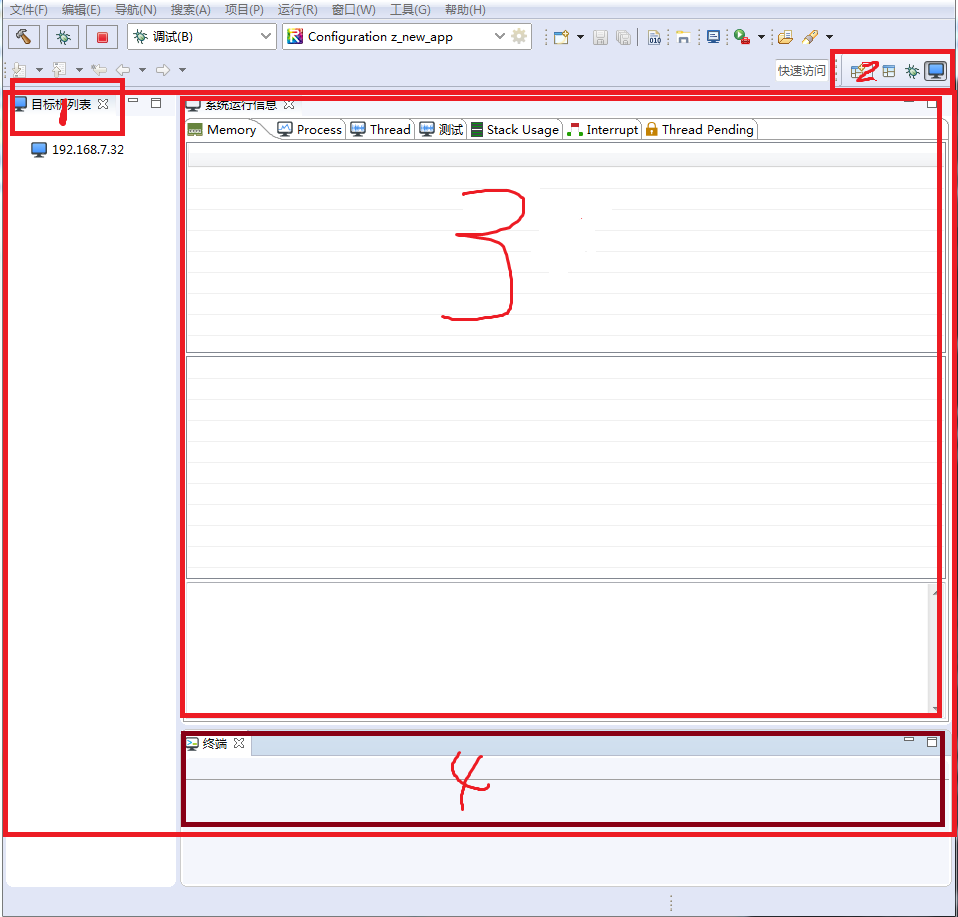
* 相关节点



* 节点概要

|  |  |  |
| --- | --- | --- |
| 节点 | 关键子节点 | 说明 |
| perspectiveExtensions | perspectiveShortcut | 视图扩展和注册 |
| org.eclipse.ui.perspective | perspective | 工作台透视图位置布局 |
| org.eclipse.ui.views | category、view | 透视图具体实现 |

* 视图说明



### (透视图扩展) org.eclipse.ui.perspectiveExtensions

perspectiveExtensions节点可以将透视图注册到 [窗口->透视图->xx] 中，如果不添加，则需要到 [窗口->透视图->其他->xx] 中打开你的透视图。主要节点见下图。



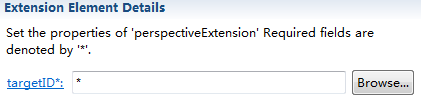
* 节点概要

|  |  |
| --- | --- |
| 关键子节点 | 说明 |
| perspectiveExtension | 透视图注册 |
| perspectiveShortcut | 透视图注册快捷方式 |

#### 透视图注册 perspectiveExtensions

透视图注册实现节点，主要进行透视图id注册，详见 **参数说明**

* 参数截图



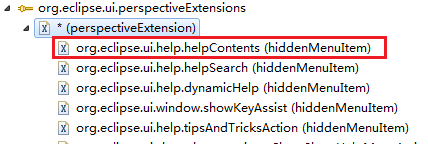
* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| targetID\* | 目标Id，需要注册的透视图ID，即*org.eclipse.ui.perspectives*的id，注册后会在窗口->透视图->打开透视图 中添加你的视图，否则只会需要在其他中寻找 |

##### 隐藏菜单栏（菜单） hiddenMenuItem

此节点可对IDE的菜单栏进行简单隐藏处理，隐藏不需要的其他菜单，实现视图的简介和功能化区分。详见**参数说明**

* 节点关系图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 需要隐藏的菜单id |

##### 隐藏菜工具条 hiddenToolBarItem

此节点可对IDE的工具条进行简单隐藏处理，隐藏不需要的其他工具，实现视图的简介和功能化区分。详见**参数说明**

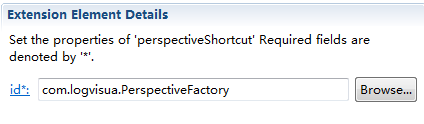
* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 需要隐藏的工具条id |

#### 快捷菜单注册 perspectiveShortcut

透视图快捷菜单注册，添加注册的透视图到菜单项中，详见 **参数说明**

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | **org.eclipse.ui.perspectives**的子节点perspective的id，通过此id将透视图添加到[*窗口->透视图*]中，不然就只能到下级菜单：其他中寻找你的视图 |

### 透视图(工作台位置布局) org.eclipse.ui.perspectives\*

perspectives节点可是实现透视图子组件的简单预布局等配配置，详见 **参数说明**

* 打开方式:

窗口->透视图->透视图名称（或在其他中寻找创建的透视图名称）只有打开此透视图，透视图按钮才会出现在 图 2 标处，view中定义的视图才会按设计的思路布局，否则通过 菜单栏：窗口>显示视图> 视图名 打开的视图，只会出现在默认的底部区域（在控制台等视图的默认区域）。

* 节点概要

|  |  |
| --- | --- |
| 关键子节点 | 说明 |
| perspective | 负责管理需要实现的具体视图的对应Layout，初始化透视图 |

#### 子节点:perspective

perspective是一个透视图管理类节点，负责管理需要实现的具体视图的对应Layout，初始化透视图。如例图所示。1、3、4位置是独立的子view，并在节点[**org.eclipse.ui.views**]\*中实现。所有子View都通过2实现处按钮实现快速切换功能。

* 关键参数：

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 建议和class名一样 |
| name\* | 透视图名称，即图 标2 处的控件名称 |
| class\* | 实现类 |
| icon | 建议在工程目录下建立一个 [icons] 文件夹，存储所有图标文件。建议像素大小为16x16，如：icons/device.png |

* Demo - class的实现:

package com.xxx;  
   
import org.eclipse.ui.IFolderLayout;  
import org.eclipse.ui.IPageLayout;  
import org.eclipse.ui.IPerspectiveFactory;  
import org.eclipse.ui.wizards.newresource.BasicNewFileResourceWizard;  
import org.eclipse.ui.wizards.newresource.BasicNewFolderResourceWizard;  
   
public class DemoPerspectives implements IPerspectiveFactory {  
   
 public Perspectives() {  
 super();  
 }  
   
 @Override  
 public void createInitialLayout(IPageLayout layout) {  
 defineLayout(layout); // 增加透视图的布局和视图  
 // defineActions(layout); // 增加透视图的动作和快捷  
 }  
   
 // 增加透视图的布局和视图  
 public void defineLayout(IPageLayout paramIPageLayout) {  
 /\* paramIPageLayout 是方位布局，即上中下左右\*/  
   
 // 设置设置编辑器区域可见不可见  
 paramIPageLayout.setEditorAreaVisible(false);  
 // 获取编辑器区域  
 String strEditorArea = paramIPageLayout.getEditorArea();  
   
 /\* 子view 布局 \*/  
 // 左边 区域 布局  
 IFolderLayout leftFolderLayout = paramIPageLayout.createFolder("leftFolder", 1, 0.18F,  
 strEditorArea); // left  
 // 添加子view的节点 id (这里是此插件节点的布局，而不是class的类名)  
 paramIPageLayout.addView("com.acce.hyide.product.gadget.navigator.view", 3, 0.65F,  
 "leftFolder"); // left-bottom  
   
 // 底部 区域布局  
 IFolderLayout bottomIFolderLayout = paramIPageLayout.createFolder("bottomFolder", 4, 0.75F,  
 strEditorArea); // bottom  
 // 添加子view的节点 id  
 bottomIFolderLayout.addView("com.acce.hyide.product.gadget.terminal.view");  
 bottomIFolderLayout.addView("com.acce.hyide.product.gadget.test.view");  
   
 // 中间区域 布局  
 IFolderLayout middleIFolderLayout = paramIPageLayout.createFolder("MiddleFolder", 2, 0.75F,  
 strEditorArea); // Middle  
 // 添加子view的节点 id  
 middleIFolderLayout.addView("com.acce.hyide.product.gadget.information.view");  
 middleIFolderLayout.addView("com.acce.hyide.product.gadget.ftp.view");  
   
 }  
   
 // 增加透视图的动作和快捷  
 public void defineActions(IPageLayout layout) {  
   
 }  
   
}

### 视图实现 org.eclipse.ui.views

views是具体视图实现配置节点，详见 **参数说明**

* 节点关系



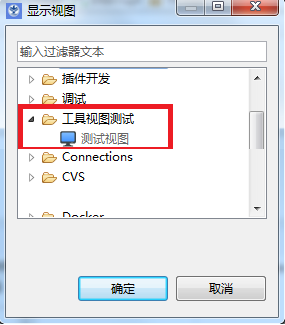
* 参数说明

#### 节点概要

|  |  |
| --- | --- |
| 关键子节点 | 说明 |
| category | 分类文件夹 |
| view | 实现具体视图的节点，被category分类 |

* 视图说明

打开方式： 菜单栏：窗口>显示视图>其他>category名



#### 节点关系

category 是分类文件夹 ，view 是视图实现节点，被category分类



#### 节点解析

##### 子节点 category：

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 分类id |
| name\* | 分类文件夹名称 |

##### 子节点 view：

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 建议为class名 |
| name\* | 子视图名称 |
| class\* | 实现类名，如com.xxxx.DemoRemoteSystemInfoView |
| category | 分类category的id |
| icon: | 图标 |

###### Demo - class实现类

DemoRemoteSystemInfoView.java

package com.xxx;  
   
import java.io.File;  
import java.text.DecimalFormat;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.List;  
   
import org.eclipse.jface.dialogs.MessageDialog;  
import org.eclipse.jface.viewers.TableViewer;  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.custom.CTabFolder;  
import org.eclipse.swt.custom.CTabItem;  
import org.eclipse.swt.custom.SashForm;  
import org.eclipse.swt.custom.StyleRange;  
import org.eclipse.swt.custom.StyledText;  
import org.eclipse.swt.events.MenuEvent;  
import org.eclipse.swt.events.MenuListener;  
import org.eclipse.swt.events.SelectionAdapter;  
import org.eclipse.swt.events.SelectionEvent;  
import org.eclipse.swt.events.SelectionListener;  
import org.eclipse.swt.layout.GridData;  
import org.eclipse.swt.layout.GridLayout;  
import org.eclipse.swt.widgets.Button;  
import org.eclipse.swt.widgets.Combo;  
import org.eclipse.swt.widgets.Composite;  
import org.eclipse.swt.widgets.Control;  
import org.eclipse.swt.widgets.DirectoryDialog;  
import org.eclipse.swt.widgets.Display;  
import org.eclipse.swt.widgets.Event;  
import org.eclipse.swt.widgets.Group;  
import org.eclipse.swt.widgets.Listener;  
import org.eclipse.swt.widgets.Menu;  
import org.eclipse.swt.widgets.MenuItem;  
import org.eclipse.swt.widgets.MessageBox;  
import org.eclipse.swt.widgets.Shell;  
import org.eclipse.swt.widgets.Table;  
import org.eclipse.swt.widgets.TableColumn;  
import org.eclipse.swt.widgets.TableItem;  
import org.eclipse.swt.widgets.Text;  
import org.eclipse.swt.widgets.Tree;  
import org.eclipse.swt.widgets.TreeItem;  
import org.eclipse.ui.part.ViewPart;  
   
public class RemoteSystemInfoView2 extends ViewPart {  
 // 分页标签的主页面  
 public CTabFolder tabFolder;  
 // 页面子标签项  
 public CTabItem tbtmNewItem;  
 public CTabItem tbtmNewItem\_1;  
 // 子标签视图  
 public TableViewer tableViewer;  
 public TableViewer tableViewer\_1;  
 public TableViewer tableViewer\_2;  
 // 标签组件  
 // 组件 ： 列表  
 public Table table;  
 public Table table\_1;  
 public Table table\_2;  
 // 组件 ： 文本框  
 public Text text;  
   
 // 布局 ：Group，可作为一个容器或页面，添加其他组件、布局等  
 public Group group;  
 // 组件 ： 下拉框  
 public Combo combo;  
 // 组件 ： 按键  
 public Button button;  
 // 组件,和 Text 差不多，和设置字体和颜色  
 public StyledText styledText;  
   
 // 树状图  
 public Tree tree;  
 private TreeItem treeItem;  
 private TreeItem treeItem2;  
 private MenuItem menuItem1;  
 private MenuItem menuItem2;  
   
 // 简单消息对话框  
 MessageBox messagebox;  
   
   
 @Override  
 public void createPartControl(Composite parent) {  
 // 容器初始化 (所有Tab页面都在此容器下)  
 tabFolder = new CTabFolder(parent, SWT.BORDER);  
 tabFolder.setSimple(false);  
   
 // 添加 UI  
 addTableView\_1();  
 addTableView\_2();  
 addGroupView\_3(parent);  
 addTreeView\_4(parent);  
   
 // 添加右键菜单  
 addMenu(table);  
   
 }  
 // 添加标签页1 方式1  
 // 布局1  
 void addTableView\_1()  
 {  
 // 标签1  
 tbtmNewItem = new CTabItem(tabFolder, SWT.NONE);  
 //tbtmNewItem.setImage(DeviceUtil.getImage("icons/Memory.png")); //$NON-NLS-1$  
 tbtmNewItem.setText("标签1");  
   
 // 标签的容器，在此容器内添加组件  
 SashForm sashForm = new SashForm(tabFolder, SWT.VERTICAL);  
 tbtmNewItem.setControl(sashForm);  
   
 // (列表视图)标签页面的组件  
 tableViewer = new TableViewer(sashForm, SWT.BORDER | SWT.FULL\_SELECTION);  
 table = tableViewer.getTable();  
 table.setHeaderVisible(true);  
 table.setLinesVisible(true);  
 // 添加列表项的监听  
 table.addListener(SWT.MouseDoubleClick, new Listener() {  
 @Override  
 public void handleEvent(org.eclipse.swt.widgets.Event event) {  
 TableItem tableitm;  
 try {  
 tableitm = table.getSelection()[0];// 获取选中的行  
 putMsg("" + tableitm.getText());  
 } catch (Exception e) {  
 return;  
 }  
 File fitem = (File) tableitm.getData();  
 if (fitem.toString().equals("..")) { //$NON-NLS-1$  
 File file = new File(strlocpath);  
 file = file.getParentFile();  
 listLocalFiles(file, table);  
 return;  
 }  
 if (fitem.isDirectory()) {  
 try {  
 listLocalFiles(fitem, table);  
 } catch (Exception e) {  
 // 遇到无权限的文件夹则返回他上级目录  
 MessageDialog.openInformation(Display.getCurrent().getActiveShell(),  
 "title",  
 "无权限的文件夹");  
 File file = fitem.getParentFile();  
 listLocalFiles(file, table);  
 return;  
 }  
 } else {  
 //if (!rimplementation.isListener) {  
 // return;  
 //}  
 //logMessage(Messages.RemoteSystemFtpImplementation\_Startuploading);  
 //rimplementation.Upload(filelist);  
 }  
 }  
 });  
   
 tableViewer\_2 = new TableViewer(sashForm, SWT.BORDER | SWT.FULL\_SELECTION);  
 table\_2 = tableViewer\_2.getTable();  
 table\_2.setToolTipText("table\_2");  
 table\_2.setHeaderVisible(false);  
 table\_2.setLinesVisible(true);  
 // (文本框视图)标签页面的组件  
 text = new Text(sashForm, SWT.BORDER | SWT.WRAP | SWT.V\_SCROLL | SWT.READ\_ONLY);  
 //text.setFont(DeviceUtil.getFont());  
 text.setBackground(Display.getCurrent().getSystemColor(SWT.COLOR\_WHITE));  
 text.setEditable(true);  
   
 // sashForm 的权重  
 sashForm.setWeights(new int[] { 161, 170, 103 });  
 }  
 // 添加标签页2 方式2  
 // 布局2  
 void addTableView\_2()  
 {  
 /\* 方式2 \*/  
 Composite compsoite = new Composite(tabFolder, SWT.NONE);  
 tbtmNewItem\_1 = new CTabItem(tabFolder, SWT.NONE);  
 //tbtmNewItem\_1.setImage(DeviceUtil.getImage("icons/Process.png"));  
 tbtmNewItem\_1.setText("标签2");  
 // 组件1  
 tableViewer\_1 = new TableViewer(compsoite, SWT.BORDER | SWT.FULL\_SELECTION);  
 table\_1 = tableViewer\_1.getTable();  
 // 组件布局方式  
 GridData gd\_tablea = new GridData(SWT.FILL, SWT.FILL, true, true, 1, 1);  
 gd\_tablea.heightHint = 220;  
 // 添加布局到 table  
 table\_1.setLayoutData(gd\_tablea);  
 table\_1.setHeaderVisible(true);  
 table\_1.setLinesVisible(true);  
 // 添加UI内容到 CTabItem  
 tbtmNewItem\_1.setControl(compsoite);  
 GridLayout layout = new GridLayout();  
 layout.marginWidth = 0;  
 layout.marginHeight = 0;  
 // 添加布局到 Composite  
 compsoite.setLayout(layout);  
   
 }  
 // 布局3 另起一组  
 void addGroupView\_3(Composite parent)  
 {  
 group = new Group(parent, SWT.NULL);  
 group.setText("Group 布局");  
 GridLayout gridLayout = new GridLayout(2, true);  
 gridLayout.marginHeight = 0;  
 group.setLayout(gridLayout);  
 group.setLayoutData(new GridData(GridData.FILL\_HORIZONTAL));  
   
 /\* 下拉框 \*/  
 combo = new Combo(group, SWT.FLAT);  
 combo.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));  
 combo.setText("默认数据");  
 combo.add("下拉数据1", 0);//添加到最前  
 combo.add("下拉数据2", 0);//不会覆盖  
 // String[] data = combo.getItems()获得所有项  
 // 为该下拉框选中事件  
 combo.addSelectionListener(new SelectionAdapter() {  
 public void widgetSelected(SelectionEvent e) {  
 String strcomnocontent = combo.getText().trim();// 选择的数据  
 MessageDialog  
 .openInformation(  
 Display.getCurrent().getActiveShell(),  
 "标题",  
 "内容:" + strcomnocontent);  
 }  
 });  
   
 /\* 按键 \*/  
 button = new Button(group, SWT.PUSH);  
 //button.setImage(DeviceUtil.getImage("icons/up.png"));  
 button.addSelectionListener(new SelectionAdapter() {  
 @Override  
 public void widgetSelected(SelectionEvent e) {  
 // 打开文件资源管理器  
 DirectoryDialog dialog = new DirectoryDialog(parent.getShell());  
 // 返回选中的文件路径  
 String strpath = dialog.open();  
 if (strpath == null) {  
 return;  
 }  
 File file = new File(strpath);  
 if (file.getParentFile() == null) {  
 //buttonUpLocalDir.setEnabled(false);  
 } else {  
 listLocalFiles(file, table);  
 putStyledText(file.toPath().toString());  
 }  
 }  
 });  
 //  
 styledText = new StyledText(group, SWT.BORDER | SWT.V\_SCROLL);  
 styledText.setSize(200, 400);  
 styledText.setEditable(false);  
 }  
   
 // 给tab页面添加右键菜单  
 // 添加 Tree 视图  
 void addTreeView\_4(Composite parent)  
 {  
 this.tree = new Tree(parent, SWT.NONE);  
 // 添加双点击事件  
 this.tree.addSelectionListener(new SelectionListener() {  
 // 双击时  
 @Override  
 public void widgetDefaultSelected(SelectionEvent e) {  
 putMsg("TreeView点击响应" + tree.getSelection()[0].getData().toString() );// 获取选中信息  
 }  
   
 // 获得(单击)焦点时  
 @Override  
 public void widgetSelected(SelectionEvent e) {  
 //putMsg("TreeView点击响应widgetSelected");  
 }  
 });  
   
 // 添加右键菜单  
 addMenu(tree);  
   
 treeItem = new TreeItem(tree, SWT.None);  
   
 treeItem.setText("subItem");  
 treeItem.setData("subItemData");  
 //treeItem.setImage(DeviceUtil.getImage("icons/connect.png")); //$NON-NLS-1$  
   
 treeItem2 = new TreeItem(treeItem, SWT.None);  
 treeItem2.setText("subItem2");  
 treeItem2.setData("subItemData2");  
 //treeItem2.setImage(DeviceUtil.getImage("icons/connect.png")); //$NON-NLS-1$  
   
 }  
   
 // 添加右键菜单 给Table、Tree等继承Control的类都行  
 private void addMenu(Control table) {  
 // 添加右键菜单  
 Menu memorymenu = new Menu(table);  
   
 // 添加操作项  
 MenuItem memoryItem = new MenuItem(memorymenu, SWT.PUSH);  
 // 操作项名称  
 memoryItem.setText("操作项1");  
 // 添加操作响应监听  
 memoryItem.addListener(SWT.Selection, new Listener() {  
 @Override  
 public void handleEvent(Event event) {  
 Thread memorymenu = new Thread() {  
 public void run() {  
 //操作响应  
 // 列表头  
 String[] strtitle = {"strtitle1","strtitle2"};  
 // 列表数据行  
 String[][] strcontent = {{"strcontent1", "strcontent2"},{"strcontent3", "strcontent4"}};  
 setTableData\_table(strtitle, strcontent);  
 }  
 };  
 memorymenu.setName("memorymenu");  
 memorymenu.start();  
 }  
 });  
   
 // 创建一个分割线  
 MenuItem separator1 = new MenuItem(memorymenu, SWT.SEPARATOR);  
   
 // 添加其他操作项  
 MenuItem connectItem = new MenuItem(memorymenu, SWT.PUSH);  
 connectItem.setText("操作项2");  
 connectItem.addListener(SWT.Selection, new Listener() {  
 @Override  
 public void handleEvent(Event event) {  
   
 }  
 });  
   
 // 菜单监听  
 memorymenu.addMenuListener(new MenuListener() {  
 // 菜单显示前  
 @Override  
 public void menuShown(MenuEvent e) {  
 // 设置操作项不可显示  
 memoryItem.setEnabled(true);  
 connectItem.setEnabled(false);  
 // 这里可以设置规则，控制操作项的显示逻辑  
 // \*\*\*  
 }  
   
 // 菜单隐藏后  
 @Override  
 public void menuHidden(MenuEvent e) {  
 }  
 });  
 table.setMenu(memorymenu);  
   
 /\* MenuItem 可重复添加 \*/  
 }  
   
 /\*\*  
 \* StyledText 显示数据  
 \* @param message  
 \*/  
 public void putStyledText(String message) {  
 styledText.getDisplay().asyncExec(new Runnable() {  
 @Override  
 public void run() {  
 StyleRange styleRange1 = new StyleRange();  
 styleRange1.start = styledText.getCharCount();  
 styleRange1.length = message.length();  
 styleRange1.fontStyle = SWT.BOLD;  
 Shell shell = new Shell();  
 styleRange1.foreground = shell.getDisplay().getSystemColor(SWT.COLOR\_RED);// Display.getCurrent().getActiveShell()  
 styledText.append(message + "\r\n"); //$NON-NLS-1$  
 styledText.setStyleRange(styleRange1);  
 styledText.setSelection(styledText.getCharCount());  
 }  
 });  
 }  
   
 public static long kb = 1024;  
 public static long mb = kb \* 1024;  
 public static long gb = mb \* 1024;  
 public static DecimalFormat df = new DecimalFormat("#,###"); //$NON-NLS-1$  
 String strlocpath = "";// 当前地址  
 /\*  
 \* 在table显示 本地路径 数据  
 \*/  
 private void listLocalFiles(File selDisk, Table table) {  
 if (selDisk == null || selDisk.isFile()) {  
 return;  
 }  
 strlocpath = selDisk.toString();  
 combo.setText(selDisk.toString());// 当前地址  
 table.removeAll();// 移除旧数据  
 File[] listFiles = selDisk.listFiles(); // 获取磁盘文件列表  
 List<File> fileList = new ArrayList<File>();  
 for (File f : listFiles) {  
 fileList.add(f);  
 }  
 Collections.sort(fileList, new Comparator<File>() {  
 @Override  
 public int compare(File o1, File o2) {  
 if (o1.isDirectory() && o2.isFile()) {  
 return -1;  
 }  
 if (o1.isFile() && o2.isDirectory()) {  
 return 1;  
 }  
 return o1.getName().compareTo(o2.getName());  
 }  
 });  
 // 上层目录  
 if (selDisk.getParentFile() != null) {  
   
 File file = new File(".."); //$NON-NLS-1$  
 TableItem itema = new TableItem(table, SWT.NONE);  
 //itema.setImage(DeviceUtil.getImage("icons/newfolder.png")); //$NON-NLS-1$  
 itema.setText(new String[] { "..", "", "" }); //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$  
 itema.setData(file);  
 } else {  
   
 }  
   
 // 遍历磁盘根文件夹的内容，添加到表格中  
 for (File file : fileList) {  
 TableItem itemsa = new TableItem(table, SWT.NONE);  
 String length = ""; // 获取文件大小 //$NON-NLS-1$  
 long lsize = file.length();  
 if (lsize >= gb) {  
 length = String.format("%.1fGB", (float) lsize / gb) + " (" + df.format(lsize) + ")"; //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$  
 } else if (lsize >= mb) {  
 float f = (float) lsize / mb;  
 length = String.format(f > 100 ? "%.0fMB" : "%.1fMB", f) + " (" + df.format(lsize) + ")"; //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$ //$NON-NLS-4$  
 } else if (lsize >= kb) {  
 float f = (float) lsize / kb;  
 length = String.format(f > 100 ? "%.0fKB" : "%.1fKB", f) + " (" + df.format(lsize) + ")"; //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$ //$NON-NLS-4$  
 } else {  
 length = String.format("%dB", lsize); //$NON-NLS-1$  
 }  
 if (file.isDirectory()) { // 显示文件夹标志  
 length = ""; //$NON-NLS-1$  
 //itemsa.setImage(DeviceUtil.getIcon(file));  
 } else {  
 //itemsa.setImage(DeviceUtil.getIcon(file));  
 }  
 // 获取文件的最后修改日期  
 //SimpleDateFormat myFmt = new SimpleDateFormat(  
 //ConstantClass.RemoteSystemFtpView\_DateFormat);  
 //String modifDate = myFmt.format(new Date(file.lastModified()));  
 itemsa.setText(new String[] { file.getName(), length, "日期" });  
 itemsa.setData(file);  
 }  
 }  
   
 // 添加 数据奥 table 列表  
 /\*添加数据到 tab 列表中\*/  
 // 给table 添加数据  
 public void setTableData\_table(String[] strtitle, String[][] strcontent) {  
 setTableData(table, tableViewer, strcontent, strtitle);  
 }  
   
 /\*\*  
 \* 给table添加信息公共方法  
 \*  
 \* @param table  
 \* @param tableViewer  
 \*/  
 void setTableData(Table table, TableViewer tableViewer, String[][] strcontent,  
 String[] strtitle) {  
 Display.getDefault().syncExec(new Runnable() {  
 @Override  
 public void run() {  
 // 记录table当前位置  
 int iscrollbarlocation = table.getTopIndex();  
 int iSelection = table.getSelectionIndex();  
 // 清空table内容  
 table.removeAll();  
   
 // 重新加载table内容  
 // if(tableViewer.getTable().getColumnCount()==0){  
 TableColumn[] columns = table.getColumns(); // 清除表格原有表头  
 for (TableColumn tc : columns) {  
 tc.dispose();  
 }  
 // 添加表头  
 for (int i = 0; i < strtitle.length; i++) {  
 TableColumn TableContent = new TableColumn(table, SWT.NONE, i);  
 TableContent.setWidth(140);  
 TableContent.setText(strtitle[i]);  
 }  
 // }  
   
 for (int j = 0; j < strcontent.length; j++) {  
 // 添加数据  
 TableItem TableTitle = new TableItem(table, SWT.NONE);  
 String strcont[] = new String[strcontent[j].length];  
 for (int k = 0; k < strcontent[j].length; k++) {  
 strcont[k] = strcontent[j][k];  
 }  
 TableTitle.setText(strcont);  
 }  
 // 设置当前选中信息位置  
 table.setTopIndex(iscrollbarlocation);  
 }  
 });  
 }  
   
 @Override  
 public void setFocus() {  
 // TODO Auto-generated method stub  
 }  
 void putMsg(String msg)  
 {  
 MessageDialog  
 .openInformation(  
 Display.getCurrent().getActiveShell(),  
 "标题",  
 msg);  
 }  
   
}

* 说明 还有部分节点未涉及，暂不说明

# 工程文件的新建和导入导出

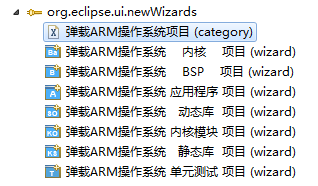
## 工程文件的新建和导入导出

通过此部分节点说明，可以自定义简单的新建文件和导入导出操作。

### 新建 org.eclipse.ui.newWizards

所有程序员的第一个项目都是通过此节点来建立的。[文件-> 新建-> ]。此节点会帮助你建立一个文件新建向导。以下是相关节点。看似不多,但部分细节实现比较负责，如需了解需查询官方文档。

* 节点关系图



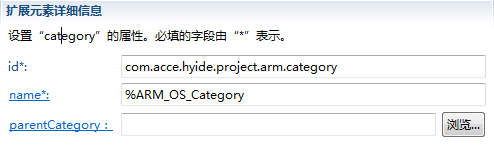
* 关键节点

|  |  |
| --- | --- |
| 节点 | 说明 |
| category | 新建项目（文件）的分类 |
| wizard | 此节点会建立一个 **新建项目（文件）向导** |

#### category 分类

此节点是一个通用节点，用来对关联节点进行分类或创建分类文件夹。详见 **参数说明**

* 参数截图



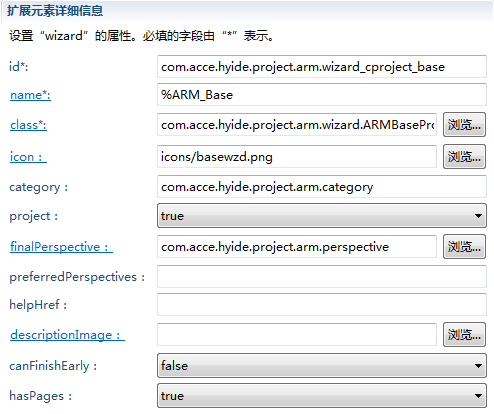
* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 分类id |
| name\* | 分类名，一般是项目工程种类名称，如新建->Java |

#### wizard

wizard节点可用于配置新建文件或新建项目的导航窗，实现wizard节点即可自定义需要的导航窗风格。节点包含的主要参数有id，class。具体内容参见**参数说明** 。

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 参数 | 说明 |
| id\* | 节点id |
| name | 新建项目（文件）类型名称,如，新建->新建txt |
| class\* | 操作实现类 |
| icon | 图标 |
| category | 新建类型分类， 如果不填，则默归类在 其他 文件夹下 |
| project | 如果为true，则新建项目工程，flase则是新建文件。 |
| finalPerspective | 最终透视图，项目所在的工作空间编译器透视图，不设置一般也没关系 |

#### wizard - calss 新建实现类

这里的demo是新建文件，如果要新建工程项目，则要继承NewCProjectWizard，系统会自动调用。

import org.eclipse.jface.viewers.IStructuredSelection;  
import org.eclipse.jface.wizard.Wizard;  
import org.eclipse.ui.INewWizard;  
import org.eclipse.ui.IWorkbench;  
   
public class SampleNewWizard extends Wizard implements INewWizard {  
   
   
 private SampleNewWizardPage page;  
 private ISelection selection;  
   
 /\*\*  
 \* 初始化  
 \*/  
 public SampleNewWizard() {  
 super();  
 setNeedsProgressMonitor(true);  
 }  
   
 /\*\*  
 \* 添加页面向导.  
 \*/  
   
 public void addPages() {  
 // 这里把 UI 单独拎出来做的 详细见demo ：SampleNewWizardPage.md  
 page = new SampleNewWizardPage(selection);  
 addPage(page);  
 }  
   
 /\*\*  
 \* 响应完成 按钮.  
 \*/  
 public boolean performFinish() {  
 final String containerName = page.getContainerName();  
 final String fileName = page.getFileName();  
 IRunnableWithProgress op = new IRunnableWithProgress() {  
 public void run(IProgressMonitor monitor) throws InvocationTargetException {  
 try {  
 doFinish(containerName, fileName, monitor);  
 } catch (CoreException e) {  
 throw new InvocationTargetException(e);  
 } finally {  
 monitor.done();  
 }  
 }  
 };  
 try {  
 getContainer().run(true, false, op);  
 } catch (InterruptedException e) {  
 return false;  
 } catch (InvocationTargetException e) {  
 Throwable realException = e.getTargetException();  
 MessageDialog.openError(getShell(), "Error", realException.getMessage());  
 return false;  
 }  
 return true;  
 }  
   
 /\*\*  
 \* 新建完成后，打开一个新编辑器显示文件  
 \*/  
   
 private void doFinish(  
 String containerName,  
 String fileName,  
 IProgressMonitor monitor)  
 throws CoreException {  
 // create a sample file  
 monitor.beginTask("Creating " + fileName, 2);  
 IWorkspaceRoot root = ResourcesPlugin.getWorkspace().getRoot();  
 IResource resource = root.findMember(new Path(containerName));  
 if (!resource.exists() || !(resource instanceof IContainer)) {  
 throwCoreException("Container \"" + containerName + "\" does not exist.");  
 }  
 IContainer container = (IContainer) resource;  
 final IFile file = container.getFile(new Path(fileName));  
 try {  
 InputStream stream = openContentStream();  
 if (file.exists()) {  
 file.setContents(stream, true, true, monitor);  
 } else {  
 file.create(stream, true, monitor);  
 }  
 stream.close();  
 } catch (IOException e) {  
 }  
 monitor.worked(1);  
 monitor.setTaskName("Opening file for editing...");  
 getShell().getDisplay().asyncExec(new Runnable() {  
 public void run() {  
 IWorkbenchPage page =  
 PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage();  
 try {  
 IDE.openEditor(page, file, true);  
 } catch (PartInitException e) {  
 }  
 }  
 });  
 monitor.worked(1);  
 }  
   
 /\*\*  
 \* 给新文件，填充初始化数据.  
 \*/  
   
 private InputStream openContentStream() {  
 String contents =  
 "This is the initial file contents for \*.mpe file that should be word-sorted in the Preview page of the multi-page editor";  
 return new ByteArrayInputStream(contents.getBytes());  
 }  
   
 private void throwCoreException(String message) throws CoreException {  
 IStatus status =  
 new Status(IStatus.ERROR, "com.acce.hyide.product.core", IStatus.OK, message, null);  
 throw new CoreException(status);  
 }  
   
 /\*\*  
 \* 初始化 workbench  
 \*/  
 public void init(IWorkbench workbench, IStructuredSelection selection) {  
 this.selection = selection;  
 }  
   
}

### 重命名 org.eclipse.ltk.core.refactoring.renameParticipants

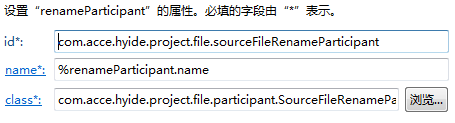
简单的节点，处理重命名操作。主要子节点有renameParticipant，详见 **参数说明** /\* 节点关系图



|  |  |
| --- | --- |
| 关键节点 | 说明 |
| renameParticipant | 处理重命名操作 |
| enablement | 操作显示逻辑，这里不做重复说明 |

#### renameParticipant 重命名处理

* 参数截图



* 参数说明

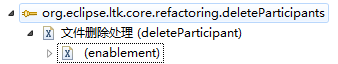
|  |  |
| --- | --- |
| 关键节点 | 说明 |
| id\* | id，一般建议为class包名 |
| name\* | 操作名称，填重命名就行 |
| class\* | 操作实现类 |

#### renameParticipant -class 操作实现类

public class SourceFileRenameParticipant extends RenameParticipant {  
   
 // 需要被删除的源文件列表  
 private List<String> deleteFilter = new ArrayList<String>();  
   
 public SourceFileRenameParticipant() {  
 // TODO 自动生成的构造函数存根  
 }  
   
 @Override  
 protected boolean initialize(Object element) {  
 if (element instanceof IResource) {  
 IResource resource = (IResource) element;  
 String projectLocation = resource.getProject().getLocation().toString();  
 if (FileUtil.underDebugOrRelease(resource.getLocation().toString(), projectLocation)) {  
 return false;  
 }  
 \*\*\*\*\*\*  
 if (!deleteFilter.isEmpty()) {  
 return true;  
 }  
 return false;  
 }  
 return false;  
 }  
   
}

### 删除 org.eclipse.ltk.core.refactoring.renameParticipants

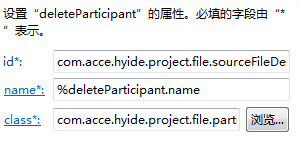
简单的节点，处理删除操作。主要子结点有 deleteParticipant，详见 **参数说明** /\* 节点关系图



|  |  |
| --- | --- |
| 关键节点 | 说明 |
| deleteParticipant | 处理重命名操作 |

#### deleteParticipant 重命名处理

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 关键节点 | 说明 |
| id\* | id，一般建议为class包名 |
| name\* | 操作名称，填重命名就行 |
| class\* | 操作实现类 |

#### deleteParticipant -class 操作实现类

public class SourceFileDeleteParticipant extends DeleteParticipant {  
   
 // 需要被删除的源文件列表  
 private List<String> deleteFilter = new ArrayList<String>();  
 private ProjectCfg cfg;  
   
 @Override  
 protected boolean initialize(Object element) {  
 if (element instanceof IResource) {  
 IResource resource = (IResource) element;  
 String projectLocation = resource.getProject().getLocation().toString();  
 if (FileUtil.underDebugOrRelease(resource.getLocation().toString(), projectLocation)) {  
 return false;  
 }  
\*\*\*  
 if (!deleteFilter.isEmpty()) {  
 return true;  
 }  
 return false;  
 }  
 return false;  
 }  
   
 @Override  
 public String getName() {  
 // TODO 自动生成的方法存根  
 return null;  
 }  
   
\*\*\*  
   
}

### 导入 org.eclipse.ui.importWizards

简单的节点，处理导入操作，主要子结点有：importWizards，详细见 **参数说明** /\* 节点关系图

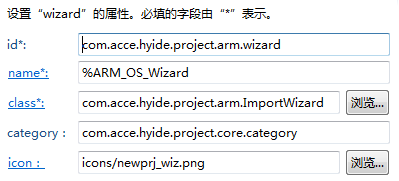


|  |  |
| --- | --- |
| 关键节点 | 说明 |
| wizards | 处理重命名操作 |

#### importWizards 导入

importWizards是负责实现导入项目配置的子结点，节点参数详见 **参数说明**

* 参数截图



* 参数说明

|  |  |
| --- | --- |
| 关键节点 | 说明 |
| id\* | id，一般建议为class包名 |
| name\* | 操作名称，填重命名就行 |
| class\* | 操作实现类 |
| category | 操作分类 |

#### deleteParticipant -class 操作实现类

import org.eclipse.jface.dialogs.IDialogSettings;  
import org.eclipse.jface.viewers.IStructuredSelection;  
import org.eclipse.jface.wizard.Wizard;  
import org.eclipse.ui.IImportWizard;  
import org.eclipse.ui.IWorkbench;  
import org.eclipse.ui.internal.ide.IDEWorkbenchPlugin;  
import org.eclipse.ui.internal.wizards.datatransfer.DataTransferMessages;  
import org.eclipse.ui.internal.wizards.datatransfer.WizardProjectsImportPage;  
   
@SuppressWarnings("restriction")  
public class ImportWizard extends Wizard implements IImportWizard {  
   
 private static final String EXTERNAL\_PROJECT\_SECTION = "ExternalProjectImportWizard";  
 private WizardProjectsImportPage mainPage;  
 private IStructuredSelection currentSelection = null;  
 private String initialPath = null;  
   
 public ImportWizard() {  
 this(null);  
 }  
   
 /\*\*  
 \* Constructor for ExternalProjectImportWizard.  
 \*  
 \* @param initialPath  
 \* Default path for wizard to import  
 \* @since 3.5  
 \*/  
 public ImportWizard(String initialPath) {  
 super();  
 this.initialPath = initialPath;  
 setNeedsProgressMonitor(true);  
 IDialogSettings workbenchSettings = IDEWorkbenchPlugin.getDefault().getDialogSettings();  
   
 IDialogSettings wizardSettings = workbenchSettings.getSection(EXTERNAL\_PROJECT\_SECTION);  
 if (wizardSettings == null) {  
 wizardSettings = workbenchSettings.addNewSection(EXTERNAL\_PROJECT\_SECTION);  
 }  
 setDialogSettings(wizardSettings);  
 }  
   
 @Override  
 public void addPages() {  
 super.addPages();  
 mainPage = new WizardProjectsImportPage(  
 "wizardExternalProjectsPage", initialPath, currentSelection); //$NON-NLS-1$  
 addPage(mainPage);  
 }  
   
 @Override  
 public void init(IWorkbench workbench, IStructuredSelection selection) {  
   
 setWindowTitle(DataTransferMessages.DataTransfer\_importTitle);  
 setDefaultPageImageDescriptor(IDEWorkbenchPlugin  
 .getIDEImageDescriptor("wizban/importproj\_wiz.png")); //$NON-NLS-1$  
 this.currentSelection = selection;  
   
 }  
   
 @Override  
 public boolean performFinish() {  
 return mainPage.createProjects();  
 }  
   
 @Override  
 public boolean performCancel() {  
 mainPage.performCancel();  
 return true;  
 }  
   
}

### 导出 org.eclipse.ui.exportWizards (暂未涉及，此处不解释)