RCP 中文入门教程

赵阳 2005.12.26

简单地讲, 一个 RCP 应用就是一个可独立于 Eclispe IDE 开发环境运行的 Eclipse 插件.

下面我们以一个简单的例子开始我们的 RCP 旅程.

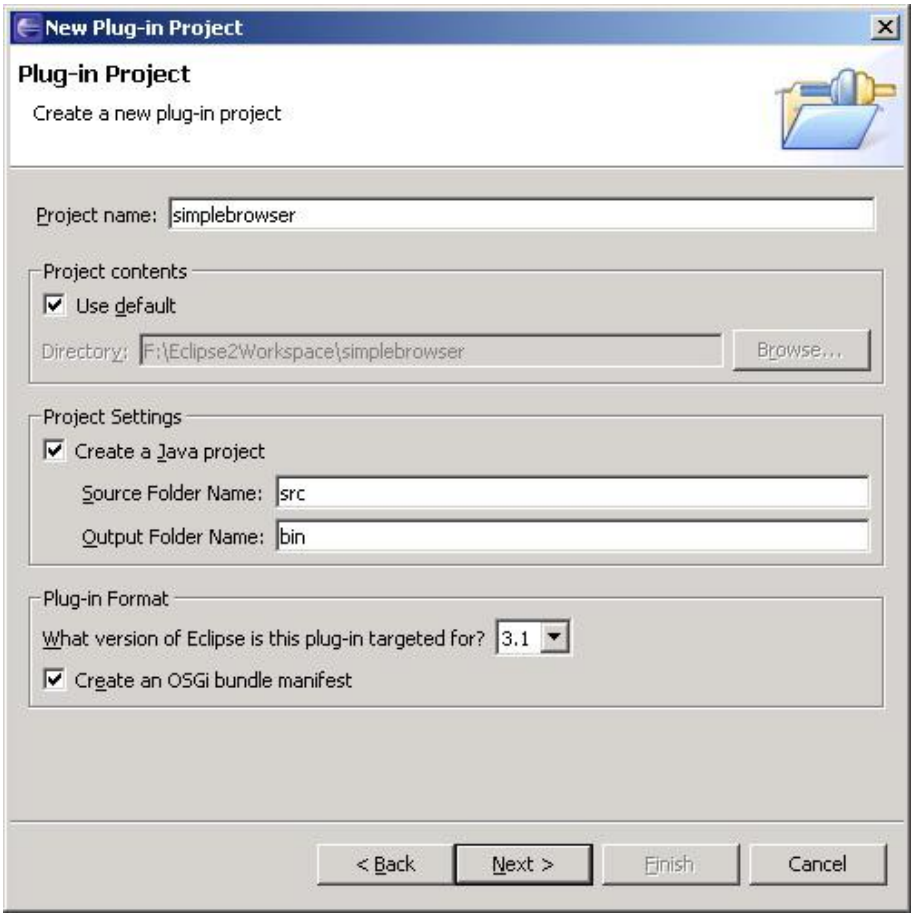
一、 新建插件工程

因为 RCP 应用本身就是一个 Eclispe 插件, 所以从新建一个 Eclispe 插件工程开始.

1) 启动 Eclispe, 从 Eclispe 的 File 菜单创建一个插件工程:

File **à** New **à** Project **à** Plug-in Development **à** Plug-in Project

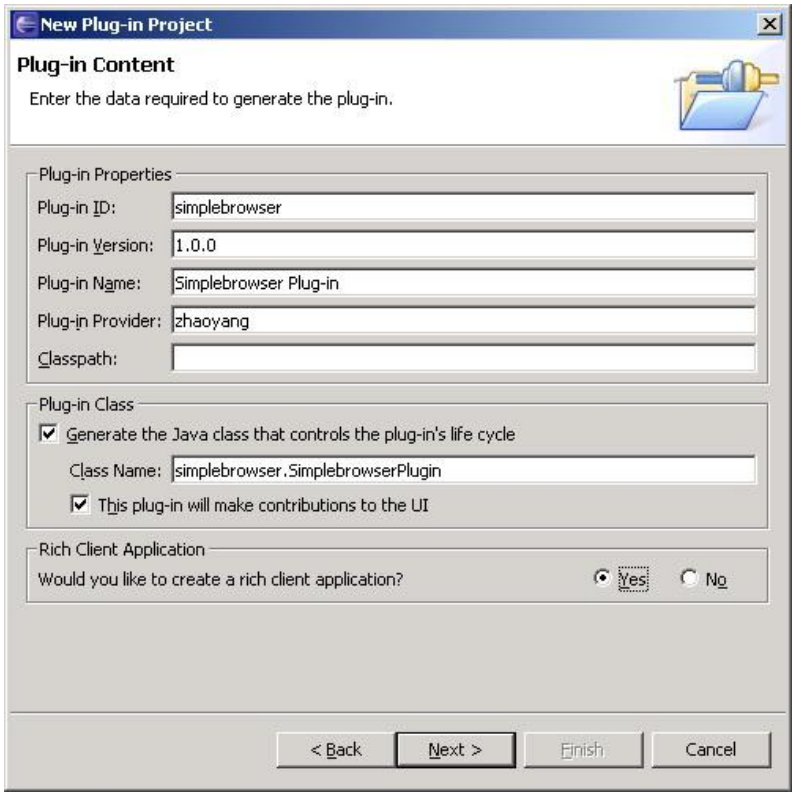
点击 Next, 进入 New Plug-in Project 插件向导:



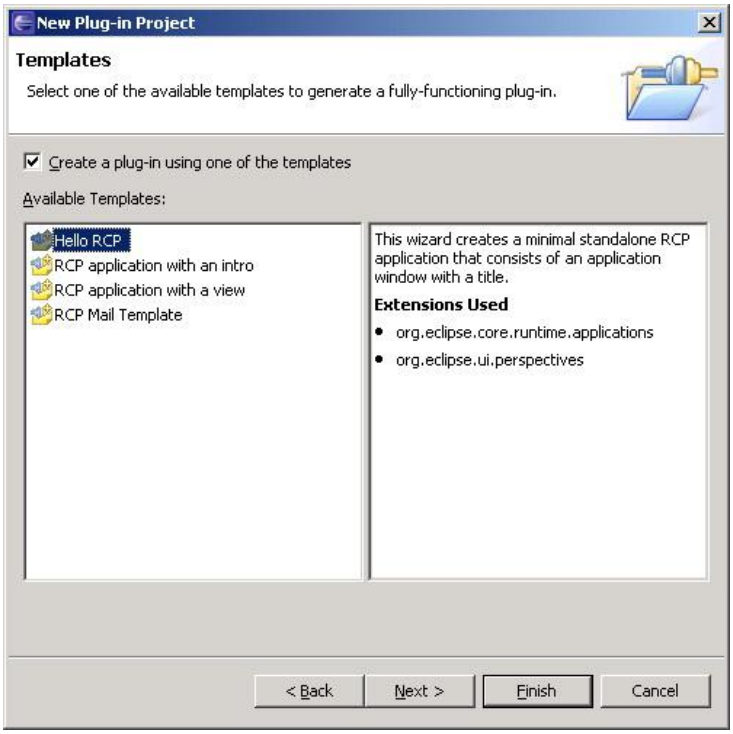
在 Project Name 中输入工程名称: simplebrowser (最好小写)

勾选 "Create an OSGi bundle mainfest" 使用 OSGi 打包清单.
OSGi 是 Eclipse 3.0 以上版本用于动态装载插件的标准, 在 Eclipse 2.1 中是不需要的. 最好选中它.

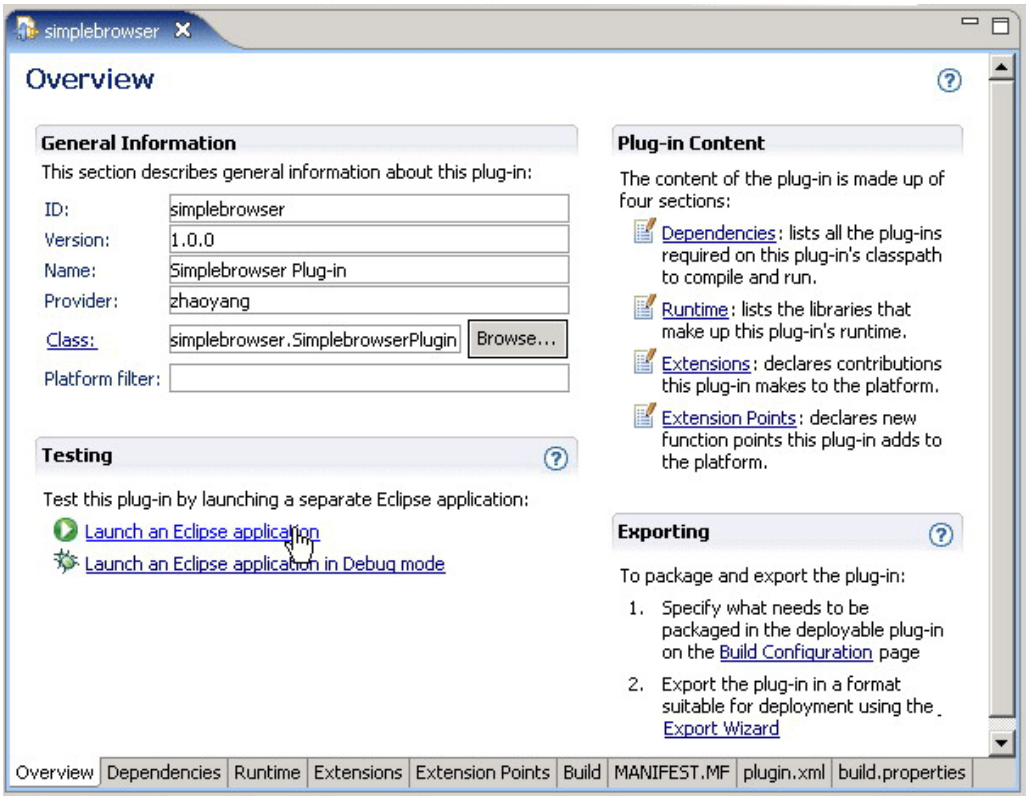
点击 Next 按钮.



修改 Plug-in ID 及其他插件属性值. 这里采用默认值, 在 ”Rich Client Application” 一栏中, “Would you like to create a rich client application ?” 一项选择 ”Yes” 设置创建的插件为 RCP 应用. 点击 Next.

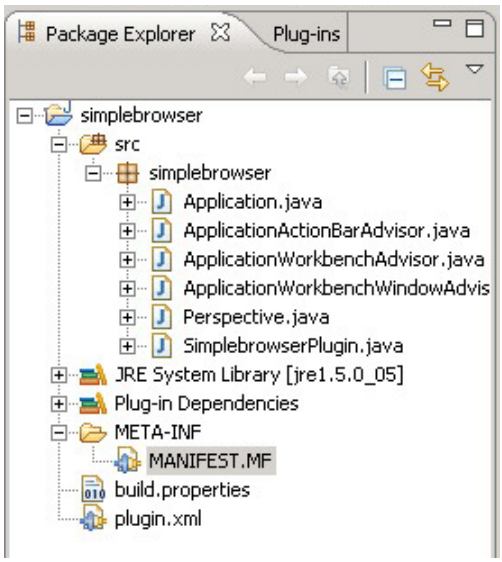


在模板中选择最基本的 Hello Rcp 模板，点击 “Finish” 按钮, Eclispe 将会创建一个简单的 RCP 应用并且自动打开插件清单编辑器的主页面. 在这里你可以方便的配置你的 RCP 应用, 免去手工编写和修改配置文件的麻烦.



在插件清单编辑器的 OverView 页, 点击 “Lanch an Eclipse application” 链接就会看到你的 RCP 应用运行时的样子.

2) 工程创建完成后, Eclipse 将自动生成以下内容:



- Application 类
- ApplicationActionBarAdvisor 类
- ApplicationWorkbenchAdvisor 类
- ApplicationWorkbenchWindowAdvisor 类
- SimplebrowserPlugin 类
- Perspective 类
- plugin.xml 文件
- build.properties 文件

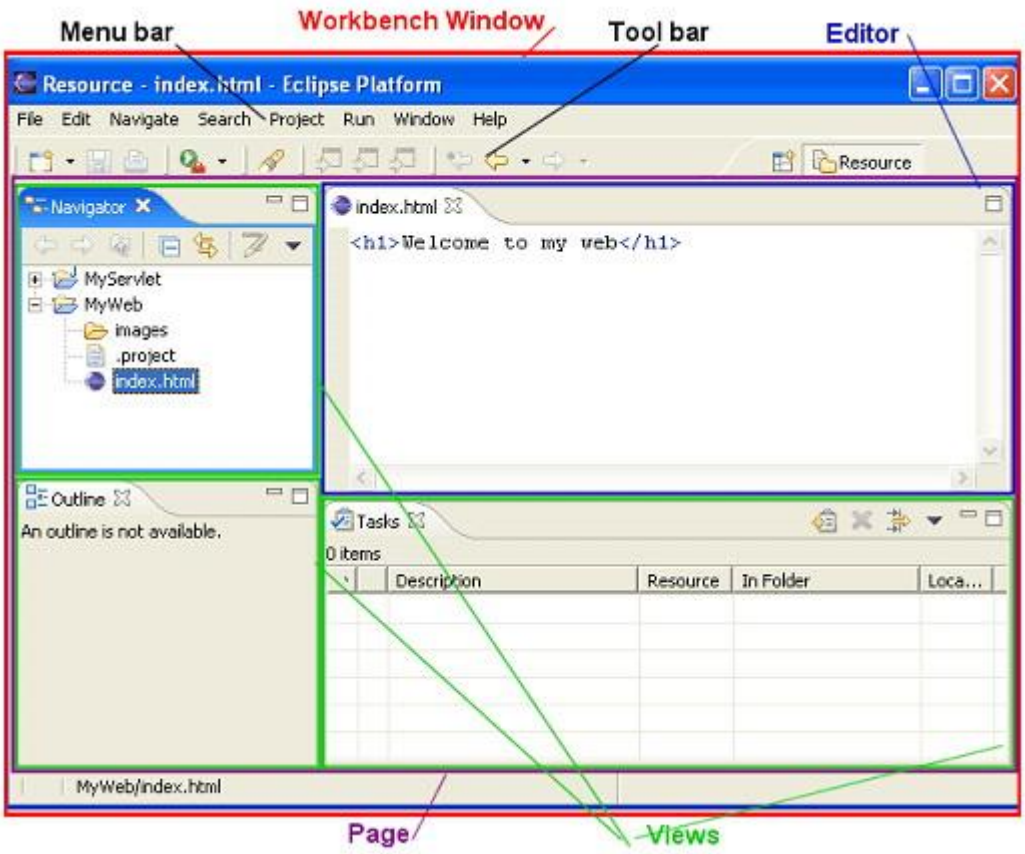
a) Application 类

Application 类是 RCP 应用的主程序, 相当于整个 RCP 应用的控制器.

Application 类的职责是创建一个工作台(Workbeanch)然后添加一个工作台顾问类(WorkbeanchAdvisior). 它是启动 RCP 应用运行的第一个程序.

这个类实现了 Eclipse 的 IPlatformRunnable 接口. 对于绝大多数的 RCP 应用, 这个类的代码都是不用修改的.

工作台(Workbeanch)是 RCP 框架的一部分, 一个 RCP 应用只能有一个工作台, 但是可以有多个工作台窗口(WorkbeanchAdvisior). 工作台的结构如下:



b) Advisor 类

ApplicationActionBarAdvisor, *ApplicationWorkbenchAdvisor* 和 *ApplicationWorkbenchWindowAdvisor* 这三个类是 RCP 应用的三个核心 Advisor 类. 它们都继承于相应的抽象 Advisor 父类. 是 RCP 应用生命周期中非常重要的三个类.

ApplicationWorkbenchAdvisor

在主程序 Application 类的 run() 方法中引用了 ApplicationWorkbenchAdvisor 类. 这个 Workbench Advisor 类对 Workbench 的外观进行了配置.

ApplicationWorkbenchAdvisor 继承了 WorkbenchWindowAdvisor 抽象类, 插件工程向导自动填充了 createWorkbenchWindowAdvisor 和 getInitialWindowPerspectiveId 方法体, 我们同样可以覆写(override)父类的其他方法. 这个类的方法会在工作台(Workbench)的生命周期的各个关键时刻由 RCP 平台调用. 是最重要的一个 Advisor 类.

下面是对 Workbench Advisor 类中几个重要的方法的简要说明:

方法	说明
initialize	在启动工作台 (Workbench) 前进行初始化. 这个方法有只有一个参数: IWorkbenchConfigurer
preStartup	在初始化完成之后, 打开第一个窗口之前调用, 在这里可以对打开编辑器和视图的初始化参数进行设置.
postStartup	在所有窗口打开或恢复以后开始事件循环之前调用. 在这里可以进行一些类似自动批处理的工作.
preShutdown	在事件循环结束以后, 关闭任何一个窗口之前调用
postShutdown	在所有窗口关闭之后, 关闭工作台 (Workbench) 之前调用, 可以用来保存当前应用的状态, 清理 initialize 方法创建的内容

现在我们不需要对这个类进行任何修改.

ApplicationWorkbenchWindowAdvisor

ApplicationWorkbenchWindowAdvisor 继承了 WorkbenchWindowAdvisor 类, 这个类主要负责对 Worbench Window 进行控制, 例如状态栏, 工具条, 标题, 窗口尺寸等.

这个类中的方法在 Workbench Window 的生命周期中起着重要作用.

方法	说明
preWindowOpen	在 WorkBench Window 的构造函数中调用, 用于设置窗口的一些特征, 如 是否显示状态栏. 但是这个时候还没有创建任何窗体控件, 所以在这里还不能引用它们.
postWindowRestore	在窗口恢复到以前保存的状态之后, 打开窗口之前调用..这个方法在新建窗口, Workbench 第一次运行, 以及没有保存窗口状态的情况下都不会调用. 在这里可以调用 *IWorkbench.close()* 方法关闭 Workbench 和所有打开的 Workbench Window.

postWindowCreate	在窗口创建以后, 打开以前调用. 或者是窗口恢复到以前保存的状态后, 在执行 *postWindowRestore* 方法之后调用.
openIntro	Intro 就是你第一次打开 Eclispe 的时候看到的内容, 这个方法的默认的实现是: 如果 *IWorkbenchPreferences . SHOW_INTRO* 属性被设置为 **True**, 那么在第一次打开窗口的时候将会调用这个方法, 在 Intro 显示过之后该属性将会设置为 **False**. 后来, 只有在 *WorkbenchConfigurer.getSaveAndRestore()* 方法返回 **True**, 并且关闭窗口时 intro 仍然显示的时候才会调用这个方法.
postWindowOpen	在 Workbench 窗口打开之后调用, 可以在这里开/关(Tweak)窗体控件, 例如设置 Title, 改变窗口尺寸等等.
preWindowShellClose	这个方法在 Workbench 窗口关闭之前 (严格的讲是它的 Shell 被关闭之前) 由关联到这个窗口的 ShellListener 调用. 如果窗口由于其他什么原因已经关闭了, 则不会调用这个方法. 如果这个方法返回 **false**, 那么关闭 Shell 的请求将会被忽略, 所以, 这个是唯一的一个可以阻止用户关闭窗口行为的地方, 也是提示用户是否保存当前工作和设置的最佳场所.
postWindowClose	在 Workbench 窗口关闭之后调用, 这个时候窗口中的控件都已经被清除了. 在这里可以清除由. *postWindowOpen* 方法创建的内容.
createWindowContents	这个方法用来创建一个窗口的内容, 默认的实现添加了一个菜单栏, 一个工具条, 一个状态栏, 一个透视图栏, 和一个快速视图栏. 这些控件的可见性可以使用 IWorkbenchWindowConfigurer 中的 setShow* 方法进行设置. 可以通过在子类中覆写(override) 这个方法来实现自定义的窗口内容和布局, 但是必须要调用 *IWorkbenchWindowConfigurer.createPageComposite*. . 方法. 这个方法只有一个参数: Shell
createEmptyWindowContents	创建并且返回在窗口没有页面显示的时候要显示的控件. 如果返回的是 **Null**, 则会使用默认的窗口背景. 覆写这个方法可以实现自定义的窗口背景. 默认的实现是返回 **Null** . 这个方法只有一个参数: Composite

插件工程向导自动帮我们填充了 createActionBarAdvisor 和 preWindowOpen 方法体, 在 preWindowOpen 方法中我们看到向导隐藏了窗口的工具条和状态栏, 并且设置了窗口的大小和标题栏上显示的文字:

```
public void preWindowOpen() {
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    configurer.setInitialSize(new Point(400, 300));
    configurer.setShowCoolBar(false);
    configurer.setShowStatusLine(false);
    // 设置窗口标题栏文字
    configurer.setTitle("Hello RCP");
}
```

在这个类的方法中常常需要使用到 Configuer 接口对窗口进行配置, 在方法体中直接调用 getWindowConfigurer()方法就可以直接获得 IWorkbenchWindowConfigurer 对象了, 就像向导在 preWindowOpen() 中所做的一样.

ApplicationActionBarAdvisor

ActionBarAdvisor 类继承了 ActionBarAdvisor 类. 这个类负责为 Workbench Window 的 Action Bar (菜单, 工具条和状态栏等) 创建 Action. 也可以通过插件清单 plugin.xml 文件动态地提供 action.

这个类中主要有下面几个方法:

方法	说明	参数
makeActions	创建在 fill 方法中使用的 action , 这个方法用来通过 key binding 服务注册 action 并且添加到关闭窗口时要清除的 action 列表中.	IWorkbenchWindow
fillMenuBar	填充窗口的主菜单	IMenuManager
fillCoolBar	填充窗口的主工具栏	ICoolBarManager
fillStatusLine	填充窗口的主状态栏	IStatusLineManager
isApplicationMenu	这个方法在使用 OLE 对象做编辑的时候使用,根据传入的菜单 ID 返回指定的菜单是应用的菜单还是 OLE 对象的菜单. 在菜单合并期间, 应当保留应用的菜单, 至于其他的菜单最好可能从 Window 中删除掉.	String

现在我们不需要对这个类进行任何修改.

以上列出来的三个 Advisor 类中的方法基本上都是不能够在客户端直接调用的, 而是由 RCP 框架掉用的. 关于这一点在相应的 API 文档中有更为详细的说明.

插件工程向导自动为我们实现了 Advisor 子类必须实现的各个方法, 根据实际需要我们可以覆写 (override) 以上各表中方法来实现我们的意图. 关于这些方法的详细资料请查询相应父类的 API 文档.

c) SimplebrowserPlugin 类

Plugin 类继承了 AbstractUIPlugin 抽象类并且是一个单例 (singleton) 类.

这个类提供了对插件 Preferences , dialog store 和 image registry 的支持, 前面两个对象提供了保存插件和对话框设置的有效途径, 后者为整个插件提供了可能在插件中频繁使用的 Image 资源 . 由此可以看出, 这个类主要用来存储插件的全局信息. 由于是单例类, 也是放置插件中其他类要用到的静态工具方法的好地方.

现在我们不需要对这个类进行任何修改.

d) Perspective 类

这是 RCP 应用的默认的透视图. 实现了 IPerspectiveFactory 接口, 并且通过 org.eclipse.ui.perspectives 扩展点的 className 属性指定.

透视图是一套可见的视图, 编辑器, 和菜单等. 在 RCP 应用中, 必须定义至少一个透视图并且设置为默认的透视图.

所有启动 RCP 应用后希望用户看到的视图或编辑器都必须在这个类的 createInitialLayout 方法中设置.

e) Plugin.xml 文件

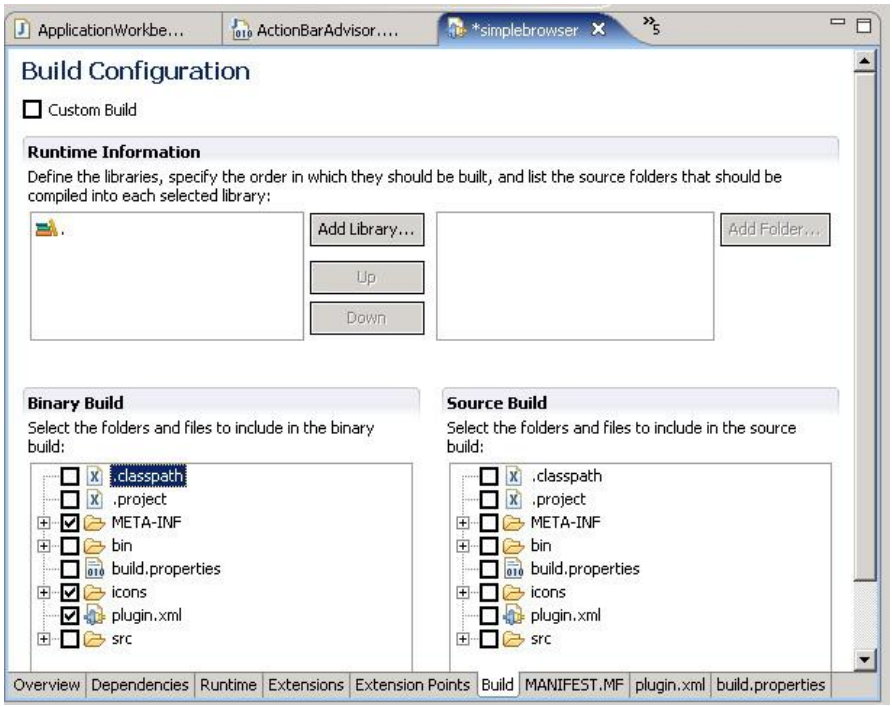
Plugin.xml 文件是 Eclipse 的扩展点清单, 这个文件用来定义和使用 Eclipse 扩展点. 扩展点是关联 Eclipse 插件的基本方式. 例如, 这里的 Application 类就是通过 org.eclipse.core.runtime.application 扩展点定义的.

Eclipse 提供了插件清单编辑器, 使用这个编辑器可以方便的修改 plugin.xml 文件的内容, 使用这个编辑器可以减少手工编辑这个 xml 文件的工作和错误. 在使用插件向导创建完工程后你看到的就是这个编辑器.

f) build.properties 文件

在导出插件的时候这个文件指定了要 build 的内容和相关资源的路径. 如果你向应用新增了一些资源, 如图标文件, 要记得把他们添加到 build.properties 文件的 bin.include 一节中.

在插件清单编辑器的 Build Configuration 页面可以很方便的在导出应用前对这个文件进行配置.



二、 添加视图

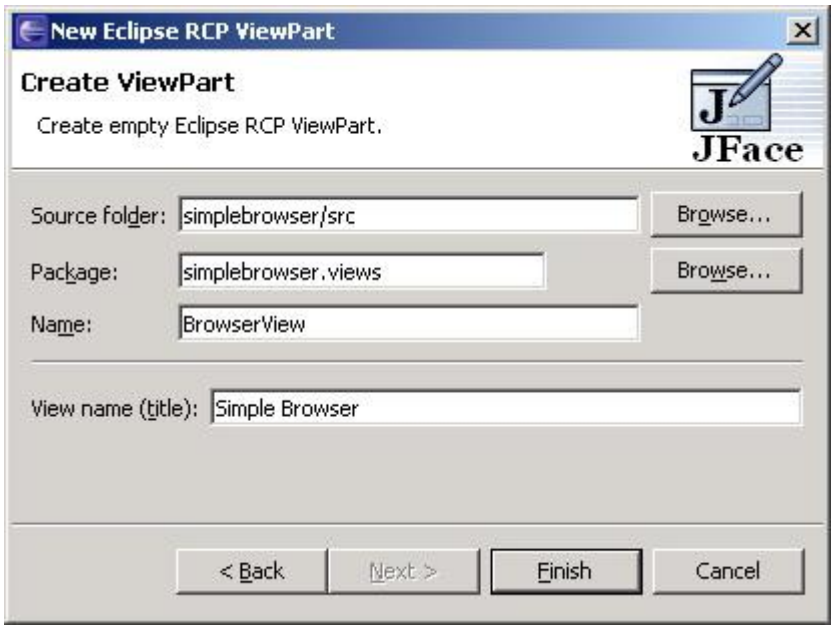
基本上 Hello RCP 模板创建的是一个空的 RCP 应用, 要实现我们自己的应用界面就需要创建我们自己的视图 (View) 并且添加到当前的透视图中. 在这里我们使用了设计器.

1) 创建视图

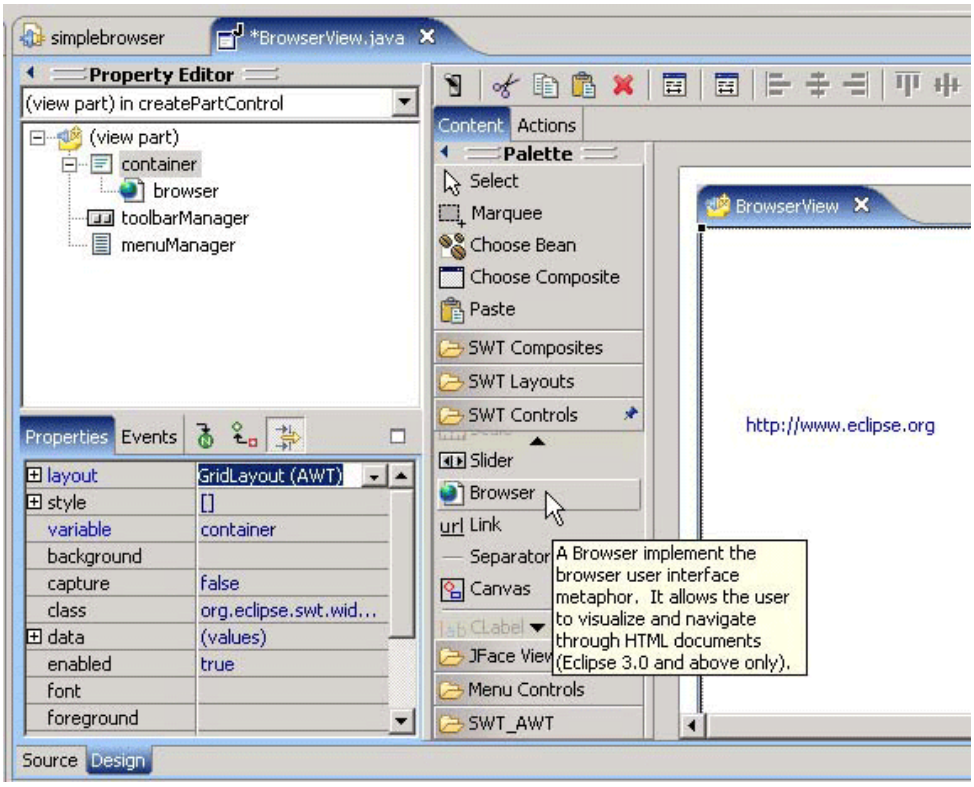
在这里我们用设计器来创建一个 RCP 视图:

在 Eclispe 中打开 RCP 插件工程, 执行以下操作:

File **à** New **à** Other **à** Designer **à** RCP **à** ViewPart **à** Next



在这里, Name 是指视图类的类名, 视图名称名称是显示在视图标题中的文字, 完成输入后点击 Finish 按钮. Eclipse 将会自动打开 BrowserView 视图类的源代码视图, 点击 Design 选项卡进入设计器视图.



把 SWT Controls 中的 Browser 控件拖到右边的 RCP 视图中, 添加 Browser 控件. 在 Property Edtor 中选中 container 在下面的 properites 页中选择 Layout 为 GridLayout(AWT), 让浏览器控件自动充满整个视图.

2) 添加视图到当前透视图中

没有添加到透视图中的视图是无法显示的.

打开 Perspective 类的源代码, 这个类是我们默认的透视图. 在它的 createInitialLayout 方法中添加以下代码:

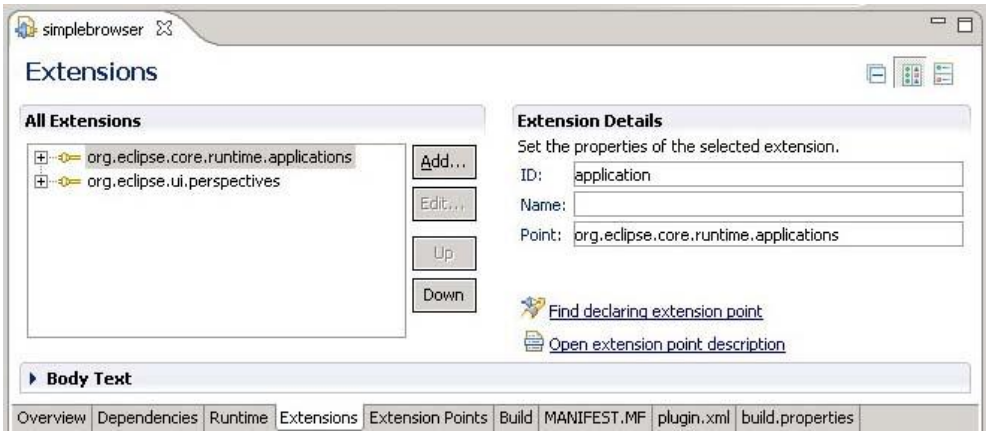
```
layout.addView( BrowserView.ID,
                IPageLayout.TOP,
                IPageLayout.RATIO_MAX,
                IPageLayout.ID_EDITOR_AREA );
```

把视图添加到当前透视图中.

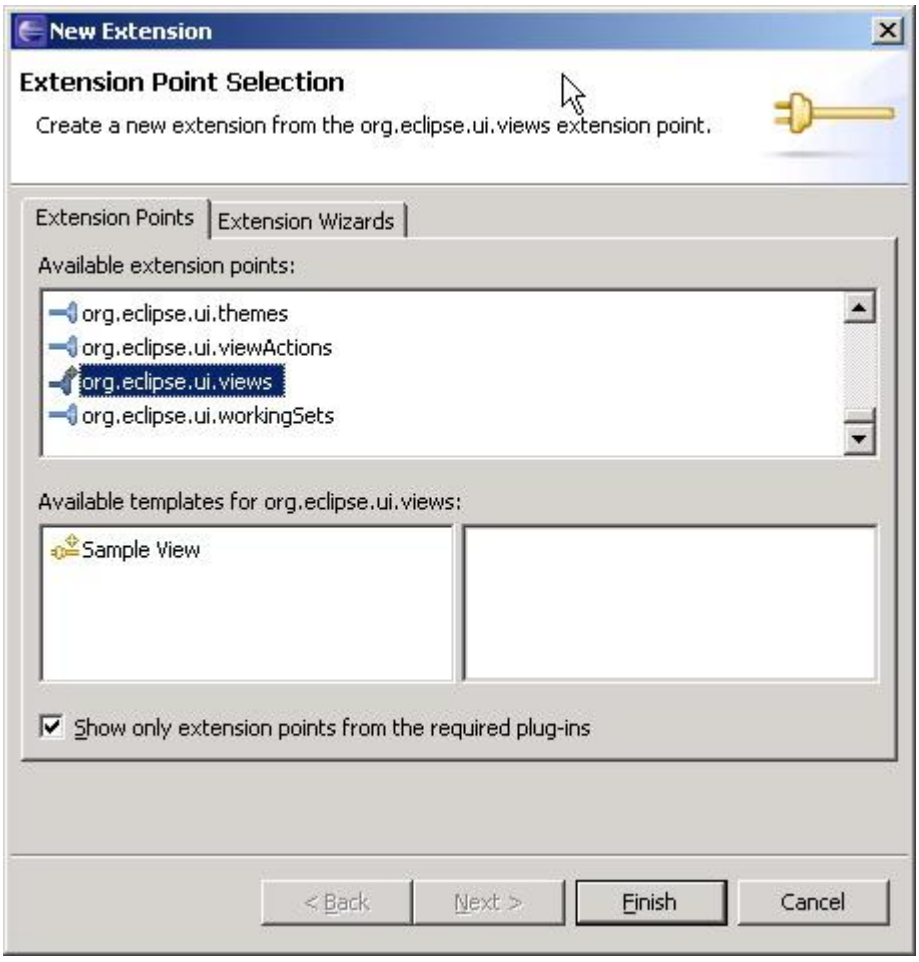
3) 添加视图的扩展点.

在刚才添加的代码中, layout 的 AddView 方法是通过视图的 ID (BrowserView.ID) 来找到我们的视图的. 这个 ID 值存储在于插件清单的 `Extension` 中, Eclipse 正是通过这些扩展点来管理插件的.

在包资源管理器 (Package Explorer) 的树形视图中, 双击 plugin.xml 文件打开插件清单编辑器, 点击 Extensions 选项卡打开扩展点清单管理页面:

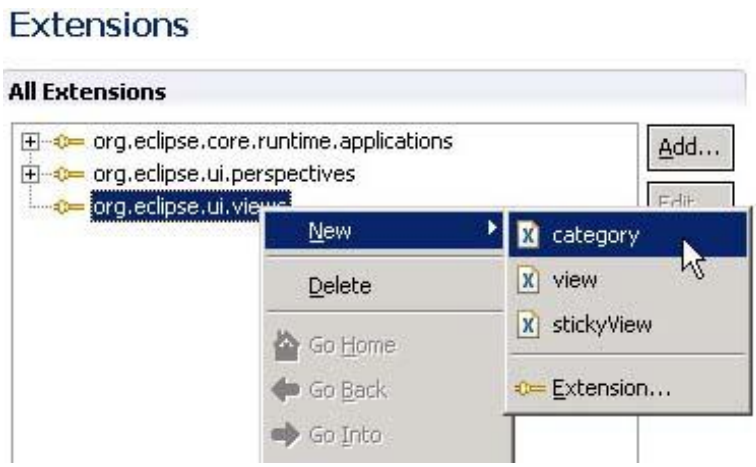


点击 Add… 按钮打开扩展点选择视图,

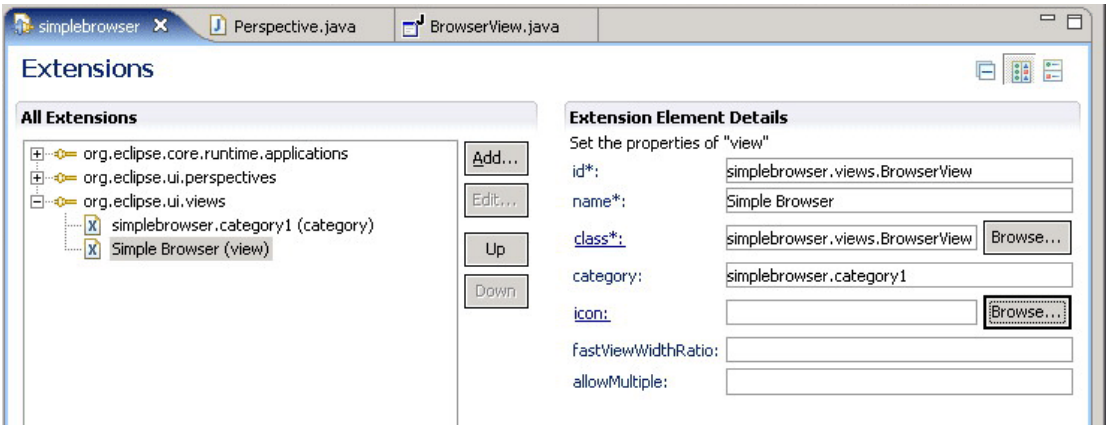


在扩展点清单中选中 org.eclipse.ui.views 扩展点, 点击 Finish 按钮, 回到插件清单编辑器, org.eclipse.ui.views 扩展点已经添加到清单中.

选中刚才添加的 org.eclipse.ui.views 单击鼠标右键, new **à** category 新建一个视图分类, 视图分类是用来分类和组织视图的, 它有三个属性: id ,name 和 parentCategory, 前两个属性用来唯一标识视图分类, 后一个属性用来组织分类的树型结构. 建议至少为你的视图添加一个分类 (Category) . 这里采用系统默认值.



以用样的方式, 选中刚才添加的 org.eclipse.ui.views 单击鼠标右键, new **à** view 添加视图:



选中刚才添加的视图, 在右边编辑视图扩展点的属性:
在 Id 一栏填写视图类中的 ID 值 : simplebrowser.views.BrowserView .
(注: 习惯上, 这个 ID 值和类的完整包路径是相同的.)
在 Name 一栏填写要显示在视图标题栏上的名称 : Simple Browser
在 Class 一栏填写视图类(含包路径的完整类名) : simplebrowser.views.BrowserView
在 icon 一栏可以指定显示在视图标题栏上的图标 (16X16 象素) .

4) 运行 RCP 应用

在 OverView 页面点击 "Lanch an Eclipse application" 链接, 你就会看到我们的视图已经显示出来了.

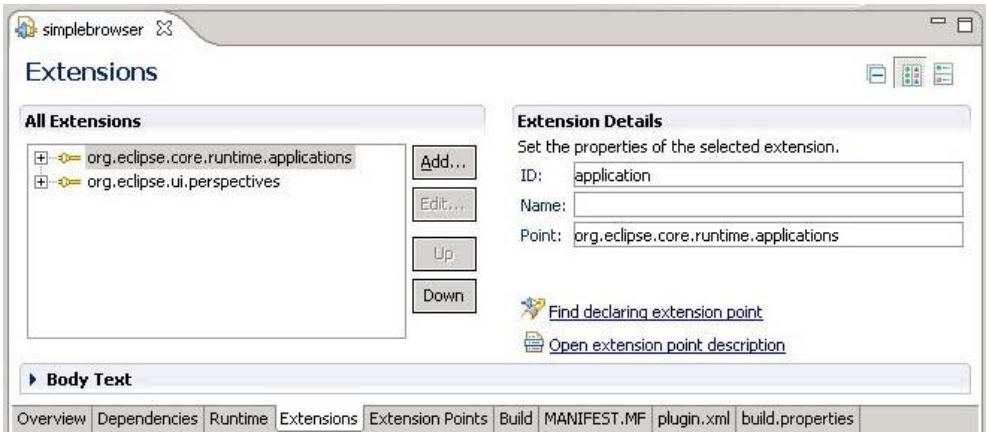
5) 手工添加视图

如果你没有设计器, 也可以通过直接继承 org.eclipse.ui.part.ViewPart 抽象类来创建视图类, 并且使用和上面相同的方法把视图添加到当前透视图中.

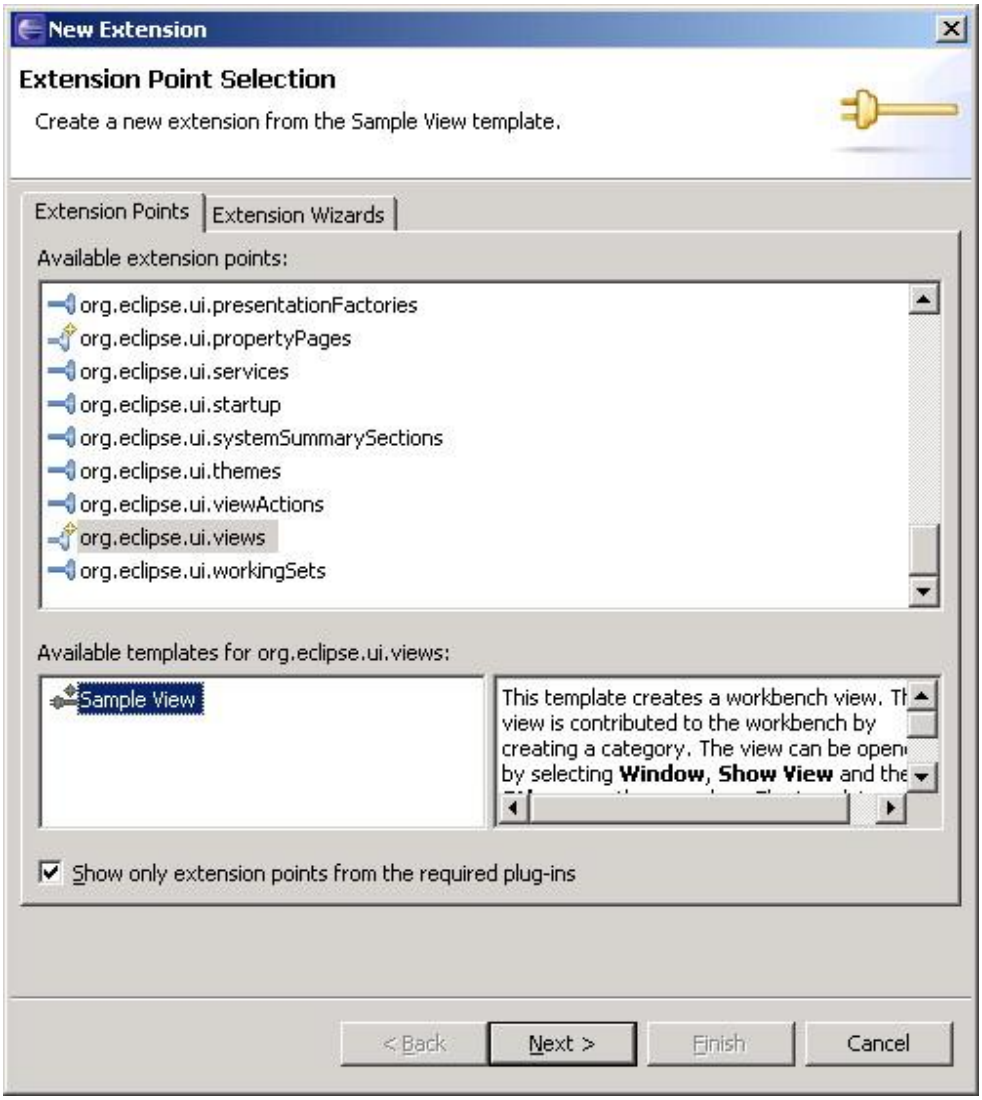
6) 使用添加扩展点向导添加视图

用上面的方式添加视图是因为设计器生成的代码比较简洁. 其实 Eclipse 提供了很好的模板和向导来创建和添加视图.

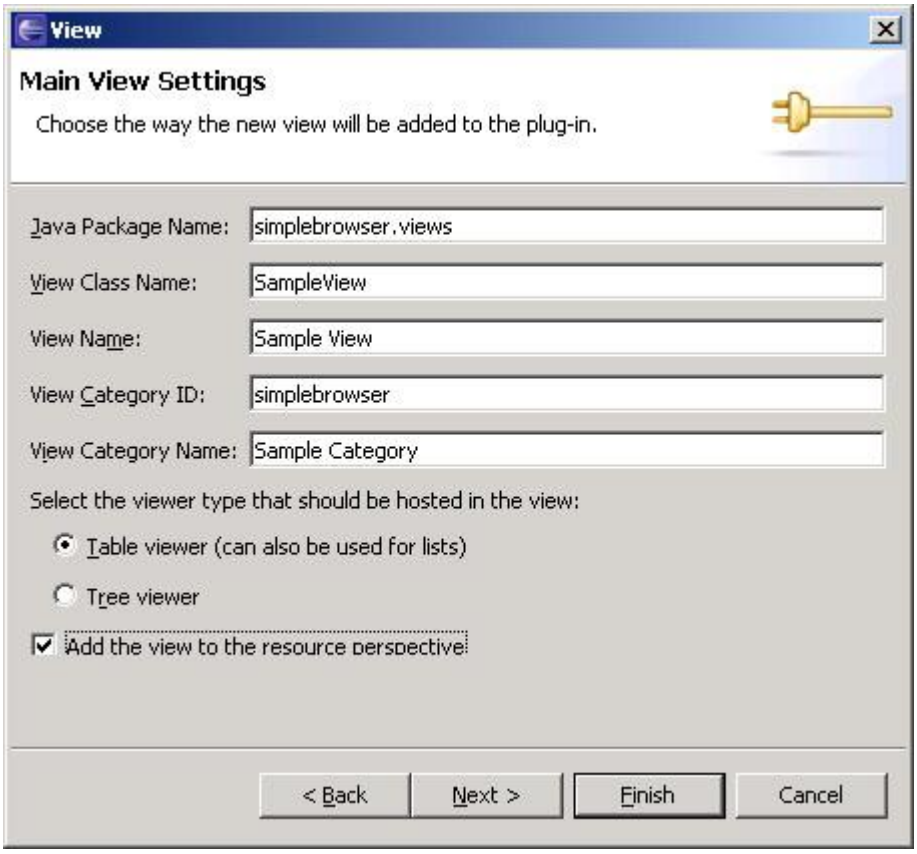
双击 Plugin.xml 文件打开插件清单编辑器, 点击 Extensions 选项卡, 进入扩展点清单页面:



点击 Add… 按钮添加扩展点:



选择 org.eclispe.ui.views 扩展点和 Sample View 模板, 点击 Next 按钮:



钩选 ” Add the view to the resource perspective ” 点击 Finish 按钮, Eclipse 将自动创建视图类并且添加相应的扩展点到插件清单中.

按照前面的方法在 Perspective 类的 createInitialLayout 方法中添加视图到透视图中, 启动 RCP 就会看到新建的视图了.

7) 一点润饰

在插件 OverView 页面点击 “Lanch an Eclipse application” 会发现透视图下方有一个空的视图区域, 这是 EditorArea, 在 Perspective 类的 createInitialLayout 方法中添加 layout.setEditorAreaVisible(false); 把它隐藏掉.

现在 Perspective 类的代码变成了这个样子:

```
public class Perspective implements IPerspectiveFactory {

    public void createInitialLayout(IPageLayout layout) {

        // 关掉 Editor Area ,
        // 如果没有用到 Editor 的话, 请执行这一句
        // 否则在透视图中会有一片空白.
        layout.setEditorAreaVisible(false);

        // 添加 Browser 视图
        layout.addView( BrowserView.ID,
                        IPageLayout.TOP,
                        IPageLayout.RATIO_MAX,
                        IPageLayout.ID_EDITOR_AREA );
```

```
        // 添加 SampleView 视图
        /*
        layout.addView( "simplebrowser.views.SampleView",
                        IPageLayout.TOP,
                        IPageLayout.RATIO_MAX,
                        IPageLayout.ID_EDITOR_AREA );
        */
    }
}
```

现在 RCP 打开的窗口的标题还是 ”Hello RCP” ,
在 ApplicationWorkbenchWindowAdvisor 类的 preWindowOpen() 方法中修改它,
并且覆写父类的 postWindowCreate() 方法在应用启动后让窗口自动最大化:
代码如下:

```
/**
 * 这个方法将在 Workbench Window 的构造函数中调用,
 * 可用来在创建窗口前设置窗口的选项:
 * 例如: 是否含有菜单栏, 状态栏等.
 */
public void preWindowOpen() {
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    configurer.setInitialSize(new Point(400, 300));
    configurer.setShowCoolBar(false);
    configurer.setShowStatusLine(false);
    // 设置窗口标题栏文字
    configurer.setTitle("Simple Browser");
}


/**
 * 这个方法在窗口恢复到以前保存的状态(或者新建一个窗口)之后,
 * 打开窗口之前(调用).
 */
public void postWindowCreate() {
    super.postWindowCreate();
    //设置打开时最大化窗口
    getWindowConfigurer().getWindow().getShell().setMaximized(true);
}
```

现在运行 RCP 应用就顺眼多了. J

三、 发布应用

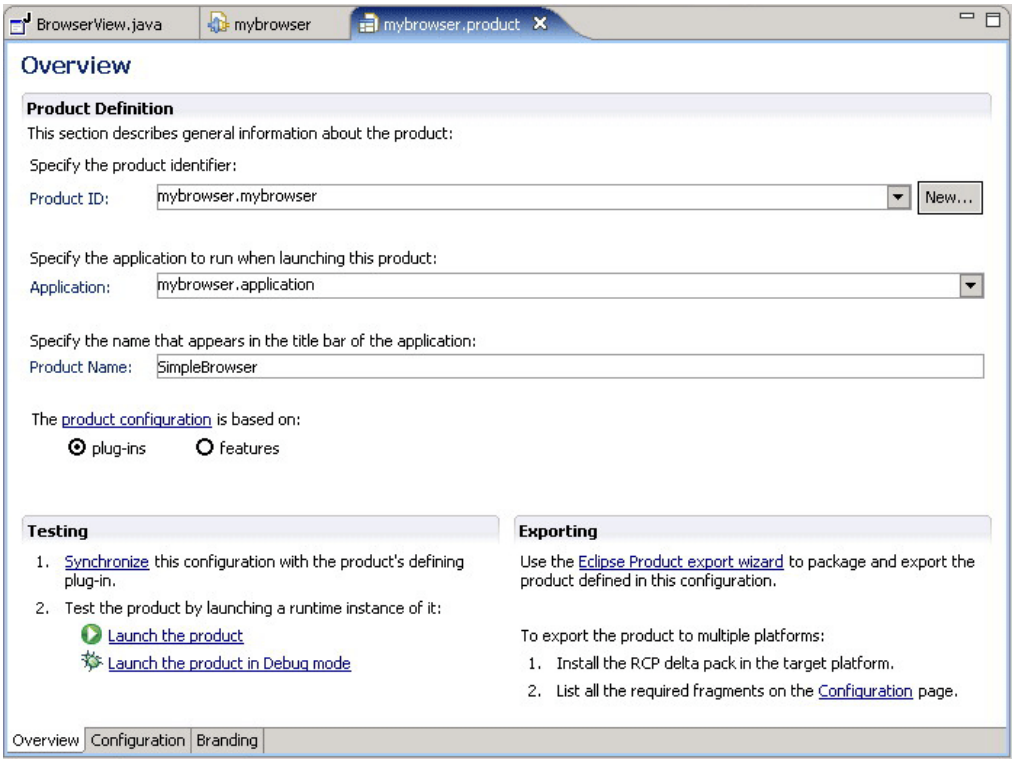
刚才建立的 RCP 应用现在还不能独立运行, 要发布一个独立的 RCP 应用, 你应当:

1) 新建 product 文件,

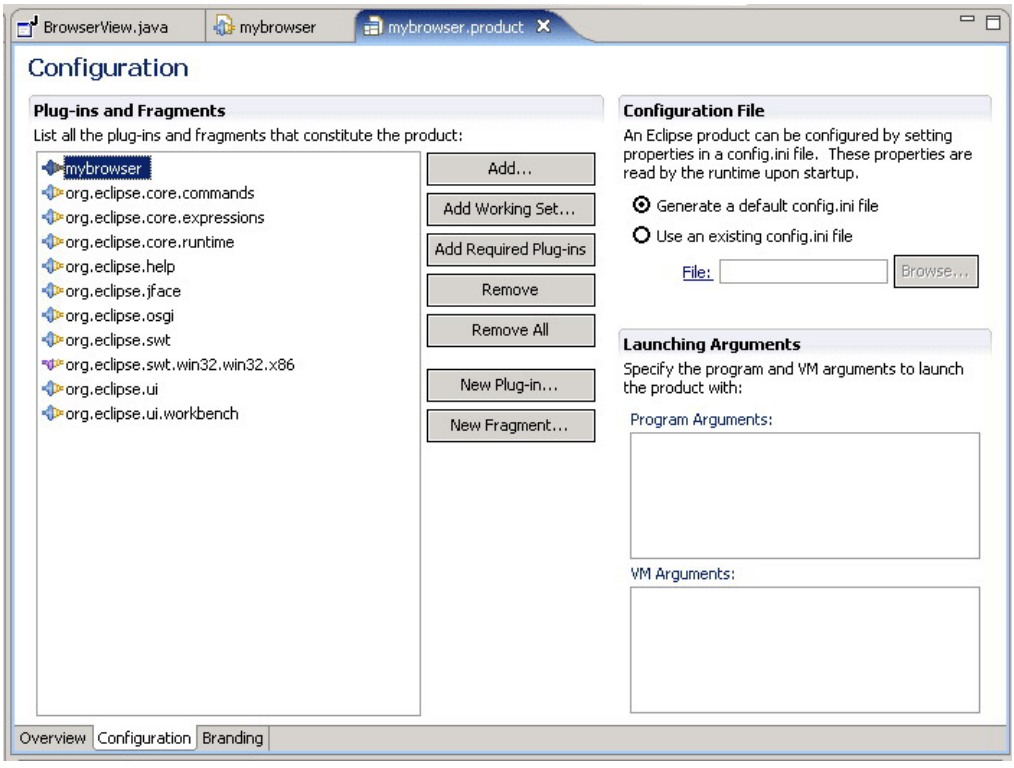
File **à** New **à** Other **à** Product Configuration **à** Next **à** 输入文件名 **à** Finish

Eclispe 将会创建一个新的 product configuration 文件并且自动打开 product configuration 编辑器.

在编辑器中进入 OverView 页面:
在 ProductID 下拉列表中选择我们创建的 RCP 插件的 ID;
在 Appliction 下拉列表中选择我们创建的 RCP 应用的 Application 主程序
在 Product Name 一栏中输入 Product 的名称.



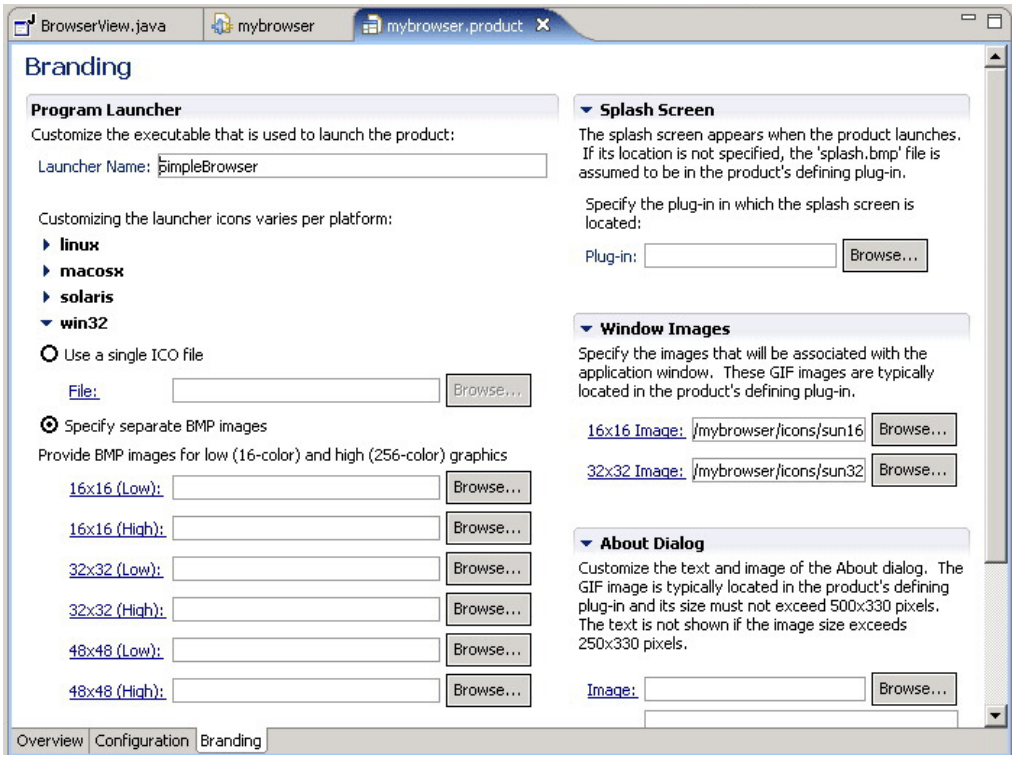
在编辑器中进入 Configuration 页面:



点击 Add… 按钮, 在弹出的列表中选中我们刚才创建的 RCP 插件: mybrowser, 点击 Ok 把我们的 RCP 插件添加到 Product 的插件列表中.

点击 Add Required Plug-ins 按钮把 RCP 应用所依赖的插件全部添加进来.

在编辑器中进入 Branding 页面:



在 Lancher Name 一栏中输入可执行程序的名称, 在 Windows 操作系统中这个名称就是你的可执行的 exe 文件的名称.

在 Window Images 一栏中选择 RCP 应用的图标. 16x16 的图标是显示在应用的标题栏上面的图标, 32x32 的图标是在切换应用程序的时候显示的图标. 如果你没有指定的话将自动放大 16x16 的图标来做显示. 建议两个都指定.

2) 导出 RCP 应用

在编辑器中进入 OverView 页面:

点击 “Eclipse Product export wizard” 链接打开 Eclispe Product 导出向导.

在 Root directory 中指定 RCP 应用的根目录,
在 Export Destination 的 Directory 中指定要导出的位置.
点击 finish 完成导出.

转到你刚才指定的导出位置, 你会发现刚才指定的 RCP 应用的根目录. 进入这个目录你就能看到在前面指定的可执行文件. 双击那个可执行的文件就 ”应该” 可以启动这个 RCP 应用了…

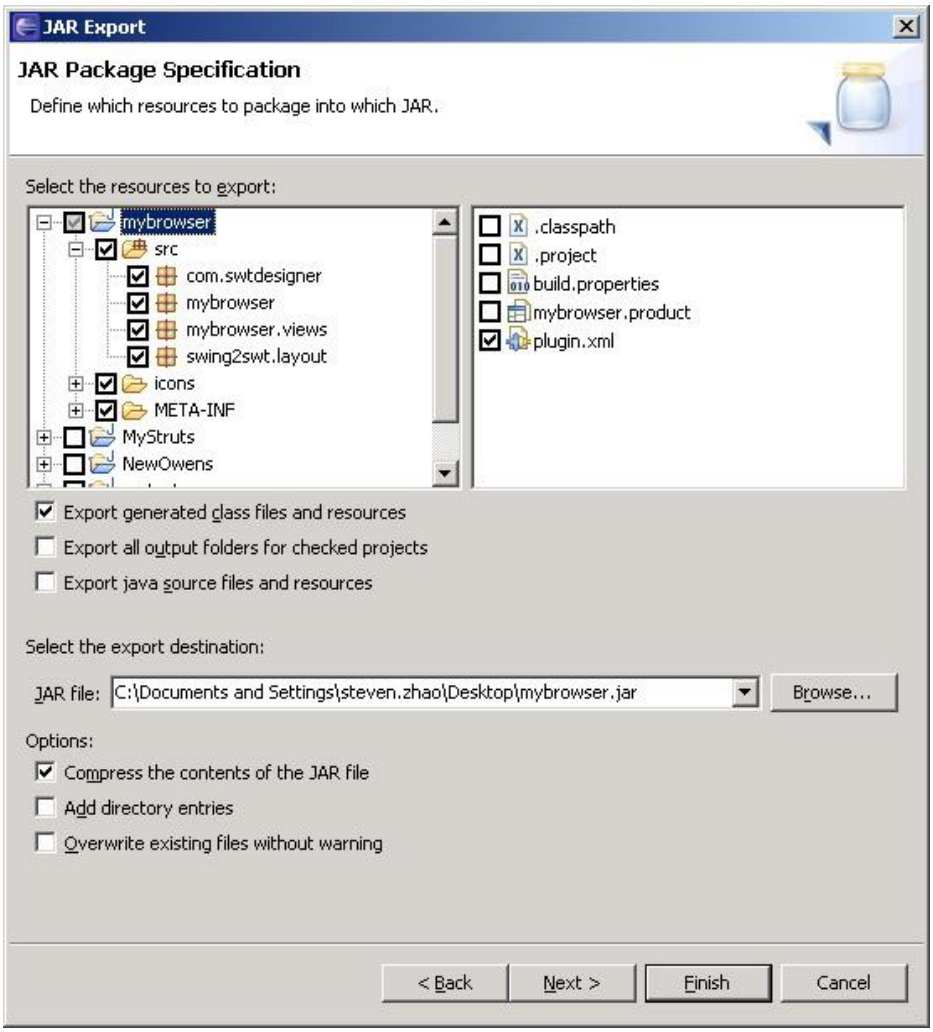
但是在启动的时候会发现产生了”Class not find” 的异常, 提示找不到 PlugIn 类.
这是由于在导出的时候生成的 Jar 包中没有包含相应的 class 文件. 关于这个问题网上有不少关于 build.properties 文件的讨论, 但是我发现下面的方法也许更加简单有效 :

3) 导出 JAR 包

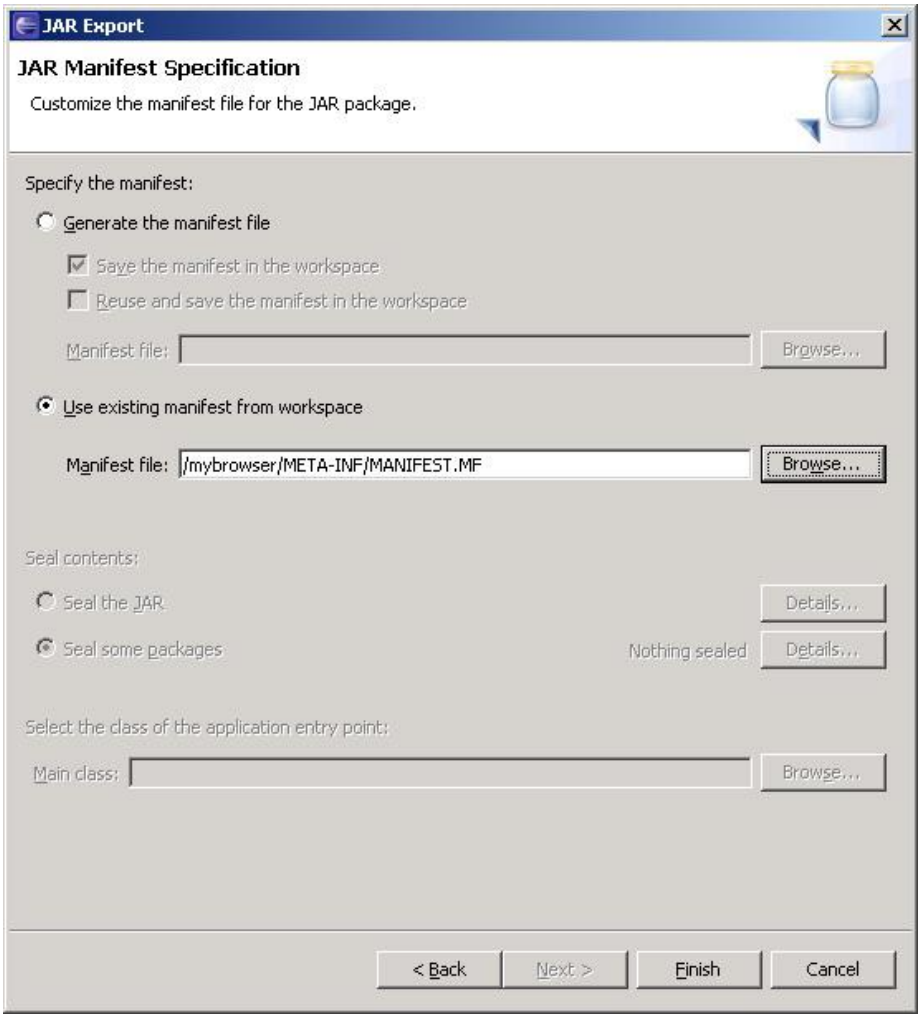
回到 Eclispe, 进行以下操作:

File à Export… à JAR file à Next

打开 JAR 文件导出向导:



在 JAR file 一栏输入要导出 Jar 文件的位置和文件名, 选中要导出的 RCP 工程, 打开 Src 目录, 只选中 plugin.xml 文件, 清除该目录下其他文件前面的选中状态, 因为这些文件在 RCP 应用运行时是不需要的. 所以不用导出. 点击 Next. 在下一个屏幕中保留默认设置, 再点击 Next, 进入 JAR 打包向导的最后一个界面.



选中"Use existing manifest from workspace" 项, 点击 Browse.. 按钮, 在弹出的对话框中, 选中 RCP 工程中 META-INF 目录下的 MANIFEST.MF 文件. 点击 OK 按钮回到向导界面. 点击 Finish 按钮, Eclipse 将会编译工程并打包到刚才指定的 Jar 文件中.

转到导出的 RCP 应用的根目录, 打开 plugins 目录, 你会发现一个包含 RCP 应用名称和版本好的 JAR 包, 如 mybrowser_1.0.0.jar . 把刚才导出的 JAR 文件重名为这个文件名. 复制到 plugins 目录中并且覆盖这个 JAR 包.

回到 RCP 应用的根目录, 双击那个可执行文件, 就会顺利的启动 RCP 应用了.

四、 参考资料

Rich Client Tutorial Part 1 (http://www.eclipse.org/articles/Article-RCP-1/tutorial1.html)

Rich Client Tutorial Part 2 (http://www.eclipse.org/articles/Article-RCP-2/tutorial2.html)

Rich Client Tutorial Part 3 (http://www.eclipse.org/articles/Article-RCP-3/tutorial3.html)