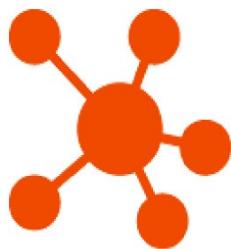


Eclipse 4 RCP 指南



目錄

1. 介绍
2. 什么是 Eclipse RCP 应用程序？
3. Eclipse 体系
4. 下载 Eclipse 软件开发包(SDK)
5. 练习：创建 RCP 应用程序向导
6. 使用运行配置
7. 启动配置及 Eclipse 产品
8. 常见的启动问题
9. Eclipse 4 应用程序模型
10. 用户接口模型元素
11. 连接模型元素到类和资源
12. 模型对象以及运行时应用程序模型
13. 特征及产品
14. 练习：建立一个插件
15. 练习：转换插件为 Eclipse 4 应用程序
16. 练习：删除持久化模型数据
17. 练习：用户接口建模
18. 练习：连接 Part 和 Java 类
19. 输入依赖
20. 练习：用 Swt 浏览器插件
21. 依赖注入介绍
22. 依赖注入与注解
23. Eclipse 上下文
24. 对依赖注入有效的对象
25. 用 Annotation 定义行为
26. 练习：用 @PostConstruct
27. 菜单以及工具条应用程序对象
28. Handler 类的依赖注入
29. @CanExecute 的执行
30. 练习：添加菜单
31. 练习：添加一个工具栏
32. 视图、弹出以及动态菜单
33. 工具栏、工具控制以及下拉工具项
34. 更多关于 commands 和 handlers
35. 核心表达式用法
36. Key 绑定
37. 通过右键启动产品
38. 关于中文翻译

基于 Eclipse 4 创建 Eclipse RCP 应用程序

本文档翻译自 <http://www.vogella.com/tutorials/EclipseRCP/article.html>

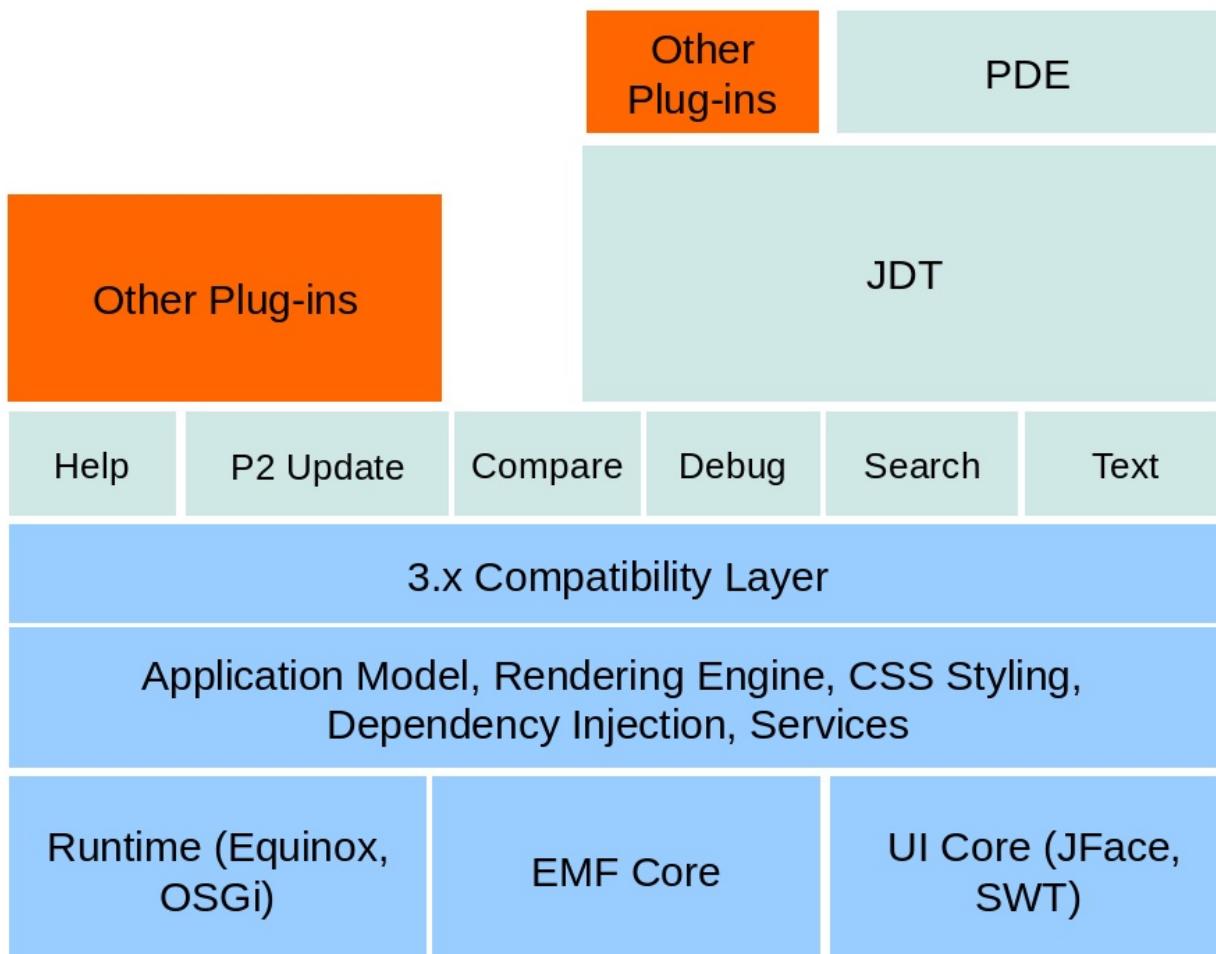
什么是 Eclipse RCP 应用程序？

一个 RCP 应用程序基于 Eclipse 平台技术的标准应用程序，本文档使用术语 Eclipse 基于的应用程序， Eclipse 应用程序， Eclipse 4 应用程序， 以及 Eclipse RCP 应用程序， 都是指这样的应用程序。

Eclipse 基于的应用程序体系

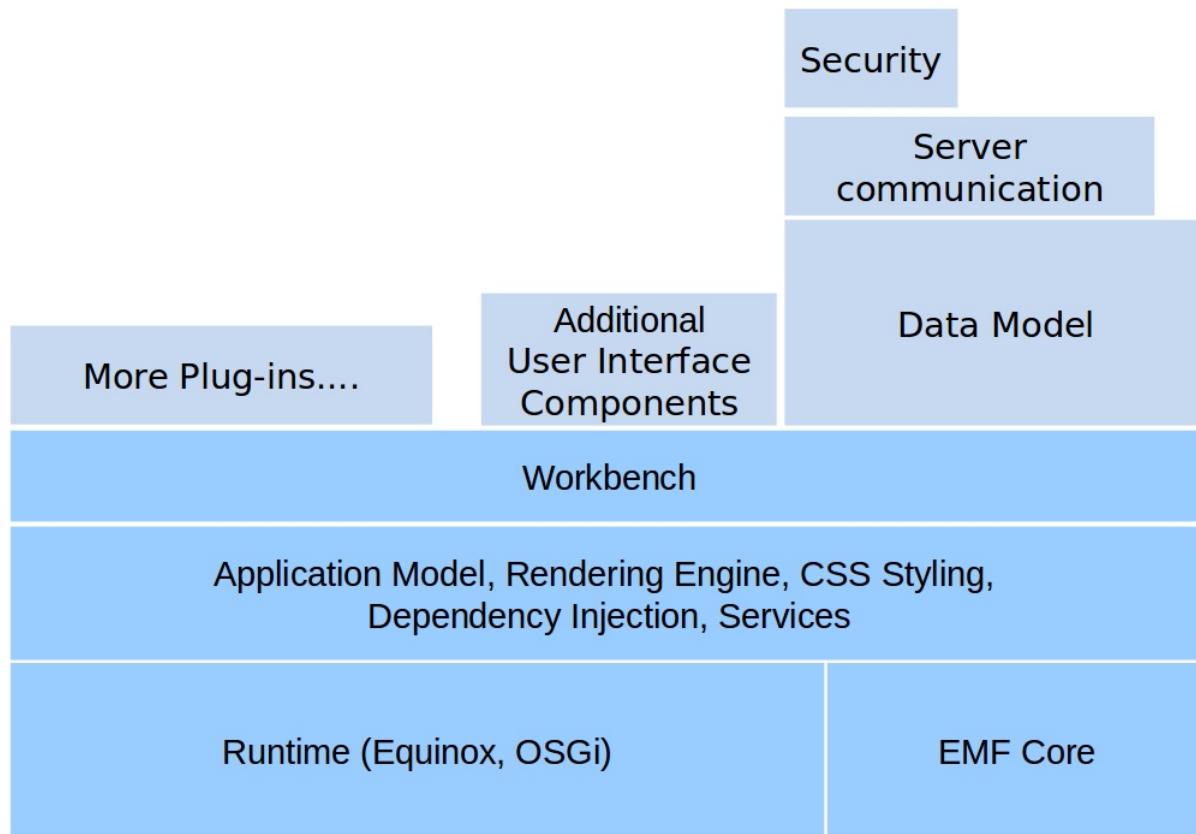
一个 Eclipse 应用程序由独立的软件组件构成，Eclipse IDE 可以被看作是特殊的 Eclipse 应用程序，其焦点是支持软件开发。

Eclipse IDE 的核心组件在下图中描述，下图主要是演示同样概念，其显示的关系不是 100% 准确的。



本图中最重要的 Eclipse 组件在后续的章节描述，在这些基础组件之上，Eclipse IDE 添加了对于 IDE 应用程序重要的额外组件，例如，Java 开发工具(JDT)以及版本控制支持 (EGIT)。

一个 Eclipse RCP 应用程序通常使用 Eclipse 平台同样的基础组件，然后添加额外的应用程序特殊的组件，如下图所示：



Eclipse 平台的核心组件

OSGi 是一个规范，描述 Java 应用程序的模块化方案，OSGi 的编程模型允许开发人员定义动态的软件组件，即 OSGi 服务。

Equinox 是 OSGi 规范的一个实现，在 Eclipse 平台中使用，Equinox 运行时提高需要的框架来运行一个模块化的 Eclipse 应用程序。

SWT 是 Eclipse 使用的标准用户接口组件库，JFace 在 SWT 之上提供一些方便的 APIs，Workbench 为应用程序提供框架，它负责显示所有其它的用户接口组件。

EMF 是 Eclipse 建模框架，提供数据模型建模的功能，然后再运行时使用这些数据模型。

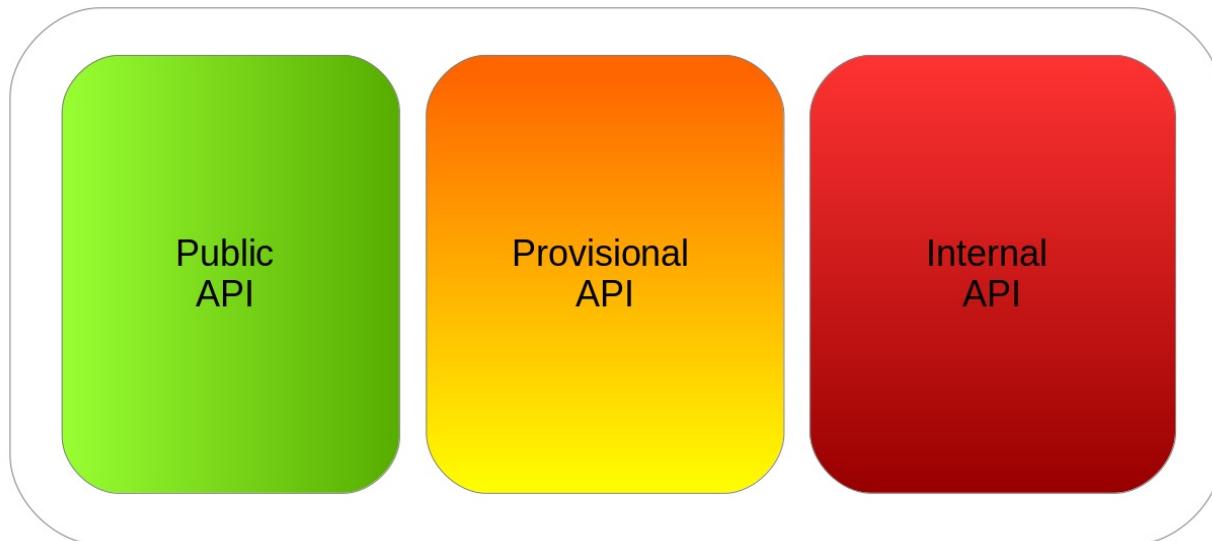
针对 Eclipse 3.x 插件的兼容布局

Eclipse 平台版本 4 采用和版本 3.x 不同的 API，大量 Eclipse IDE 已经存在的插件还是基于 Eclipse 3.x 编程模型。

Eclipse 平台提供一个兼容层，来支持使用 Eclipse 3.x API 的插件可以不加修改的在 Eclipse 4 基于的应用程序中运行。

Eclipse API 以及内部 API

OSGi 运行时允许开发人员标注 Java 包为 API 以及标注包为内部 API，也可能标注 Java 包为临时性 API。这允许开发人员测试这些 API，但指示这些 API 还没有完成。



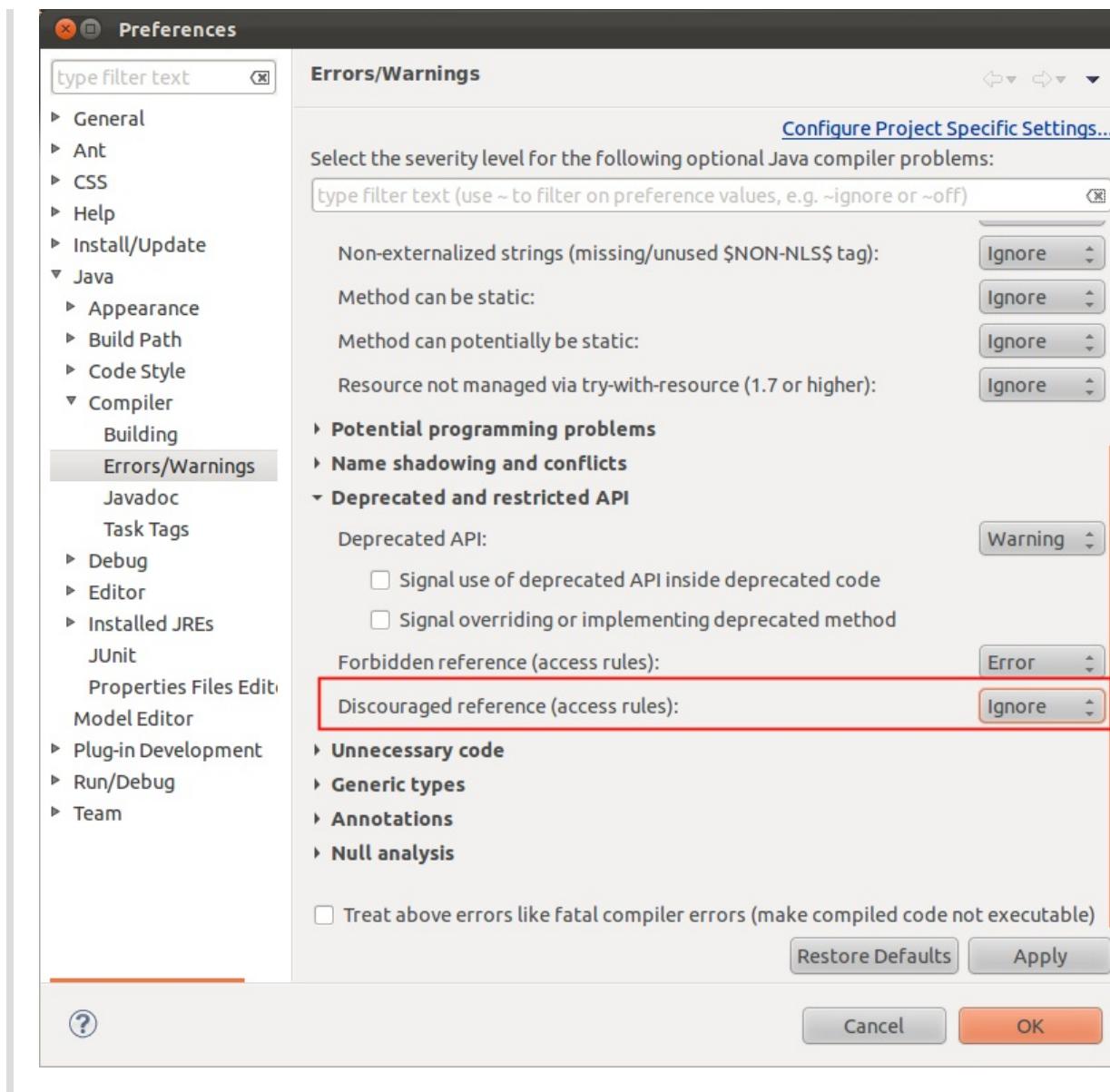
Eclipse 平台项目标注包要么为公共 API，要么为临时性 API，来让 Eclipse 开发人员可访问所有的 Java 类。如果 Eclipse 平台项目发行一个 API，平台项目会努力保持这些 API 尽可能的长期稳定。

如果 API 是内部但是可访问的，即标注为临时性的，平台团队可能在将来改变这个 API。如果你使用这样的 API，你必须准备好在将来的 Eclipse 发行时调整应用程序来适应。

如果你用非发行的 API，你会在 Java 编辑器中看到一个警告：

```
The ...is not API (restriction on required project ...)
```

提示：你可以通过 窗口 → 首选项 → Java → 编译器 → 错误/警告，然后设置 Discouraged reference(access rules) 标志为忽略，来关闭这些警告。



你也可以根据每个项目来关闭这些警告，通过右击项目，选择属性 → Java 编译器，然后用前面一样的路径。你可能必须在 Error/Warning 首选项的上方激活 允许项目特殊的设置。

Eclipse 插件重要的配置文件

一个 Eclipse 插件有下列的主要配置文件，这些文件在 API 中定义，是插件的依赖。

- MANIFEST.MF - 包含 OSGi 配置信息。
- plugin.xml - 可选的配置文件，包含关于 Eclipse 特定的扩展机制的信息。

一个 Eclipse 插件通过 MANIFEST.MF 文件定义了它的 meta 数据，例如它的唯一标识，它的输出 API 以及它的依赖。

plugin.xml 文件提供创建和贡献到 Eclipse 特定 API 的可能性。你可以在这个文件中添加扩展点和扩展。扩展点定义其它插件贡献功能的接口，扩展贡献功能到这些接口，功能可以是代码，也可能是非代码，例如插件可以只包含帮助内容。

下载 Eclipse 软件开发包(SDK)

下面的描述基于最后的 Eclipse 4.4 发行，从下面的地址下载 Eclipse SDK。

```
http://download.eclipse.org/eclipse/downloads/
```

这个网站看起来应该类似下面的截图，点击最后的发行版本链接（最高版本号的发行版本）跳转到下载页面。

Latest Release	Build Name	Build Status
4.4.2		 Ju (3 of 3 platforms)

下载文件是一个压缩包，压缩包格式根据平台不同，Windows 是 ZIP 格式，Linux 以及 Mac OS 是 tar.gz (tar包) 格式。

安装 e4 工具

注意：Eclipse 4.5 不需要额外安装 e4 工具。

从 vogella GmbH 更新站点安装 e4 工具

Eclipse SDK 下载不包含 e4 工具，e4 工具提供了创建和分析 Eclipse 4 RCP 应用程序的工具，这些工具提供向导来创建 Eclipse 应用程序模板，一个模型编辑器，以及几个分析 Eclipse 应用程序的工具。

vogella GmbH 公司提供针对 Eclipse 4.4 的 e4 工具的最近版本，在下面的地址：

<http://dl.bintray.com/vogellacompany/e4tools-luna>

你可以通过 帮助 → 安装新软件...，然后输入上面的 URL 来安装 e4 工具，来自 vogella 更新站点的 e4 工具安装包是未签名的。要安装签名的 e4 工具，采用“从 Eclipse 更新站点安装 e4 工具”中描述的方法。

Available Software

Check the items that you wish to install.

Work with: <http://dl.bintray.com/vogella/company/e4tools-luna/>

type filter text

Name
► <input checked="" type="checkbox"/> Eclipse 4 - All Spies
► <input checked="" type="checkbox"/> Eclipse 4 - Context Spy
► <input checked="" type="checkbox"/> Eclipse 4 - Core Tools
► <input checked="" type="checkbox"/> Eclipse 4 - CSS Spy
► <input checked="" type="checkbox"/> Eclipse 4 - Event Spy
► <input checked="" type="checkbox"/> Eclipse 4 - Model Spy

创建 Eclipse 4 RCP 应用程序只需求 Eclipse 4 - 核心工具。在某些特征不能安装的情况下，你可以忽略它们，继续开发 Eclipse 4 RCP 应用程序。在安装之后重新启动 Eclipse IDE。

提示：这个更新站点或它的内容可能会改变，e4 工具信息页包含关于 e4 工具更新站点的最新信息。特别是对于 Eclipse 4.5，你应该看看这个，因为这个工具计划在 4.5 中改变。

从 [Eclipse.org](http://eclipse.org/e4/tools) 更新站点安装 e4 工具

vogella GmbH 更新站点是为了提供一个使用 e4 工具的稳定的链接，同样的代码被 Eclipse.org 用来为 Eclipse e4 工具创建官方的更新站点。

不幸的是这个更新站点的链接经常改变，但是它可以在下面的网站找到：[Eclipse.org e4tools 站点](http://eclipse.org/e4/tools)。

如果你点击一个 `Build Name` 链接，你可以得到更新站点的 URL，下面的截图演示这点。

Stable Builds

Build Name

[0.17](#) ← Click here
[0.16](#)
[0.15](#)



注意这个网站可能会随着时间改变。

[Contact](#) | [Legal](#)

eclipse

- ▶ Eclipse e4
- ▶ Eclipse SDK 4.4
- ▶ Eclipse SDK 3.8

Stable Build: 0.17

201501051100. Built against Eclipse 4.3 SDK These downloads are provided under the [Eclipse Foundation Software User Agreement](#).

The page provides access to the various sections of this build along with details relating to its results. Test results are provided below and performance results are posted once they are available. You may access the download page specific to each platform by selecting one of the tabs in the platform navigator above.

A list of pre-requisite components that you need to run e4. If you install e4 from the update site, the pre-requisites will be installed automatically:
E4 runs on the [Eclipse 4.3 SDK or compatible](#).

Modeled UI and CSS

→ EMF runtime 2.9

Programming Language Support - JavaScript

→ WST SDK @wtpBuildId@ - JSDT

See E4/JavaScript for details on using the Rhino Debugging Support with Eclipse 3.6



Download now: Eclipse e4

To download a file via HTTP click on its corresponding http link below.

Related Links

- View the [compile logs](#) for the current build.
- View the [build logs](#) for the current build.
- View the [test results](#) for the current build.
- View the [map file entries](#) for the current build.

Source Builds

- Access the [Source Builds](#) page, under construction

URL for the update site

Comments

online p2 repo link

Eclipse e4		Download	Size	File	Comments
Status	Platform	(http)	15121931	eclipse-e4-repo-incubation-0.17.zip	online p2 repo link

提示：这个更新站点的内容可能会改变，e4 工具信息页包含关于 e4 工具更新站点的最新信息。尤其是对于 Eclipse 4.5，你应该检查这个站点，因为其计划在 4.5 中改变。

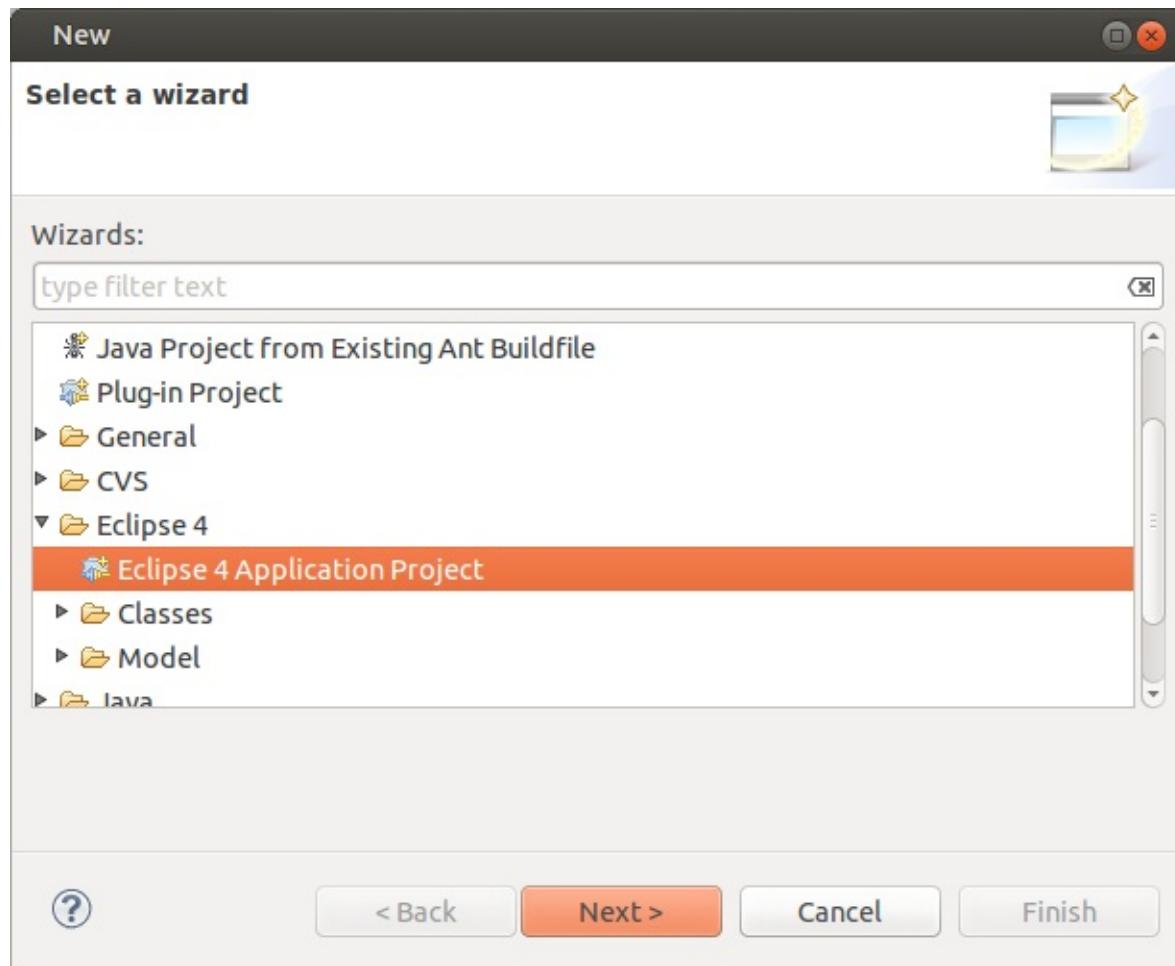
练习：创建 RCP 应用程序向导

目标

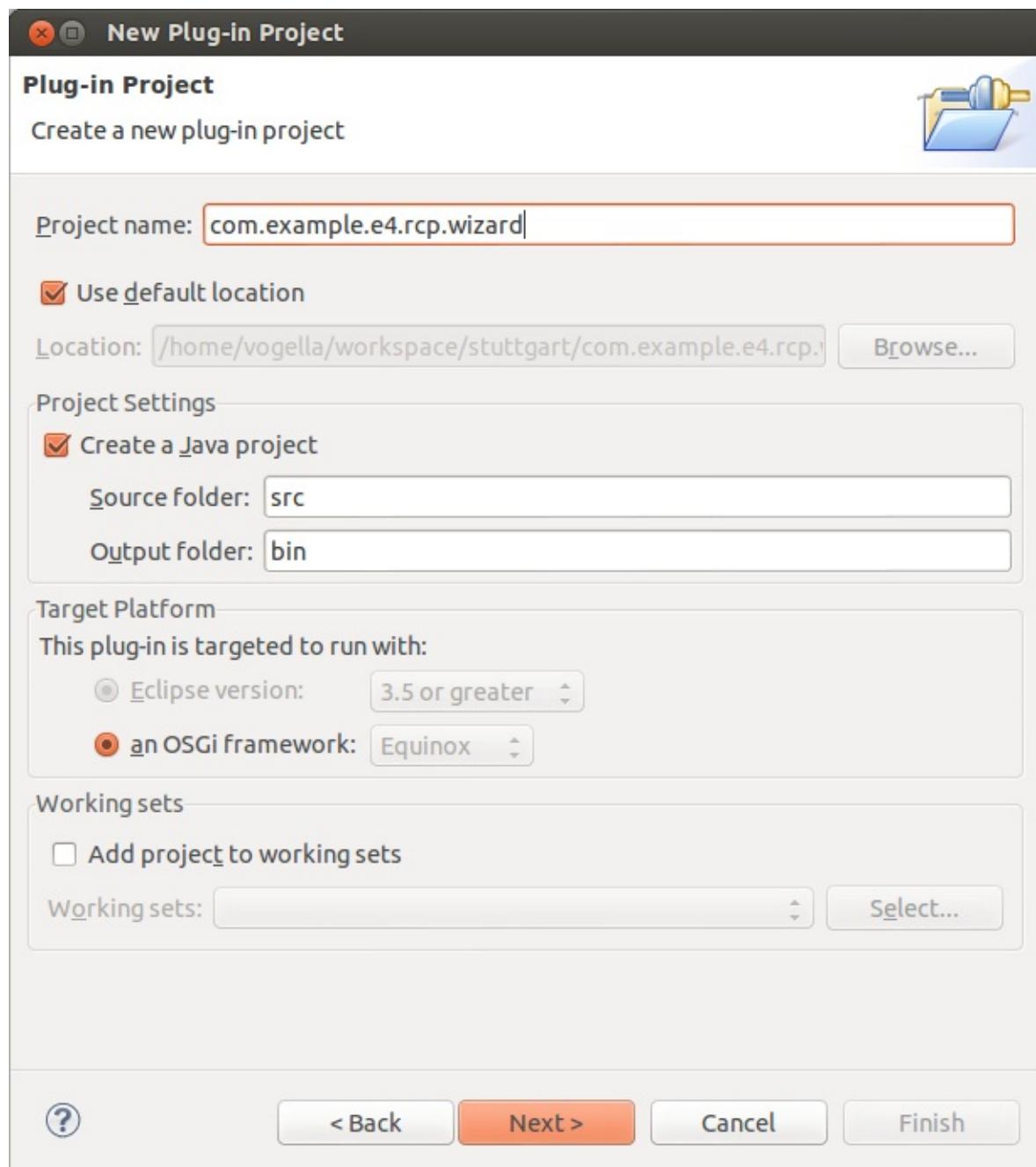
接下来的练习演示如何用 Eclipse e4 工具提供的项目创建向导来创建 Eclipse RCP 应用程序，它也演示如何通过 Eclipse IDE 启动应用程序。

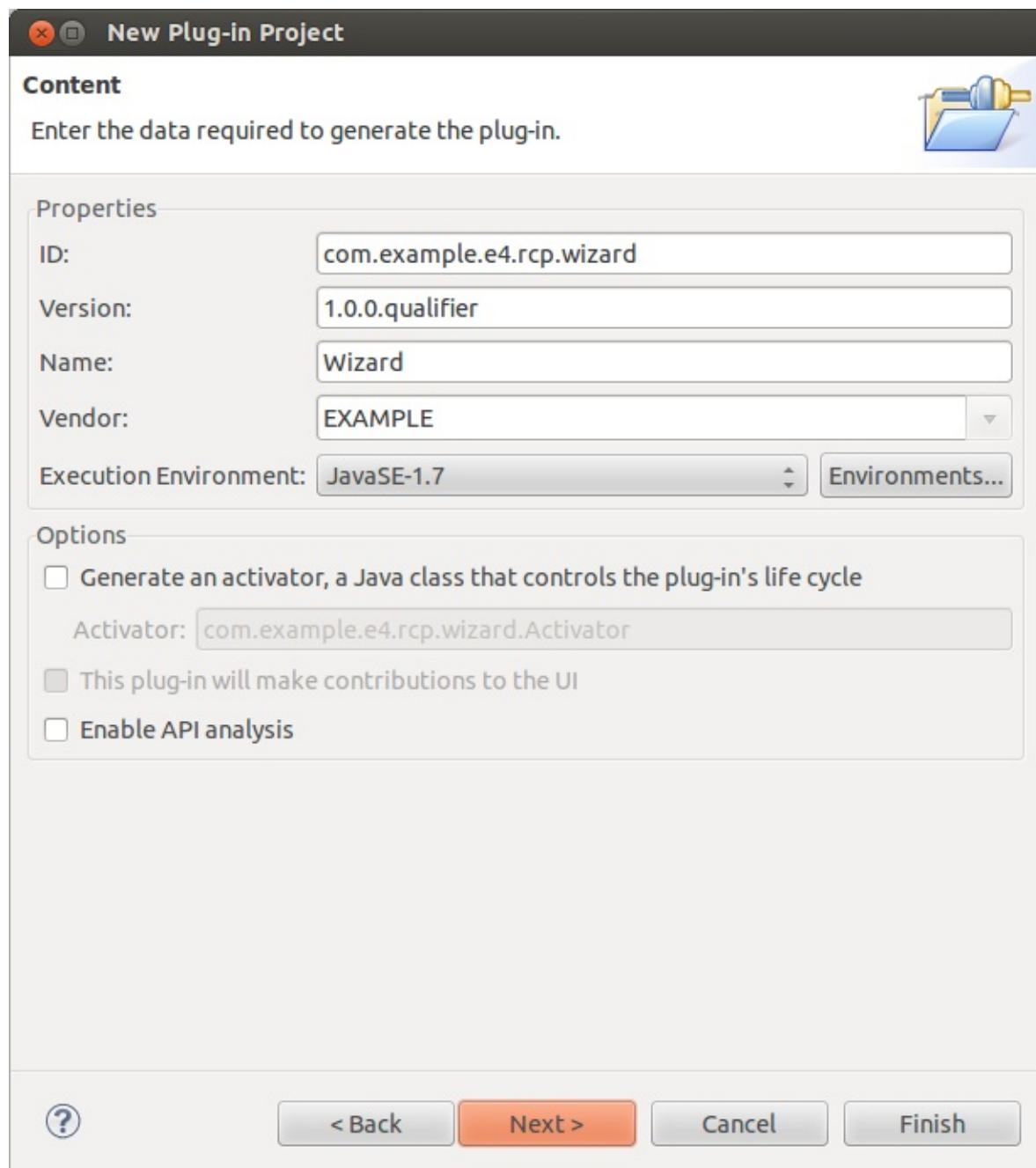
创建项目

选择文件 → 新建 → 其它... → Eclipse 4 → Eclipse 4 应用程序项目。

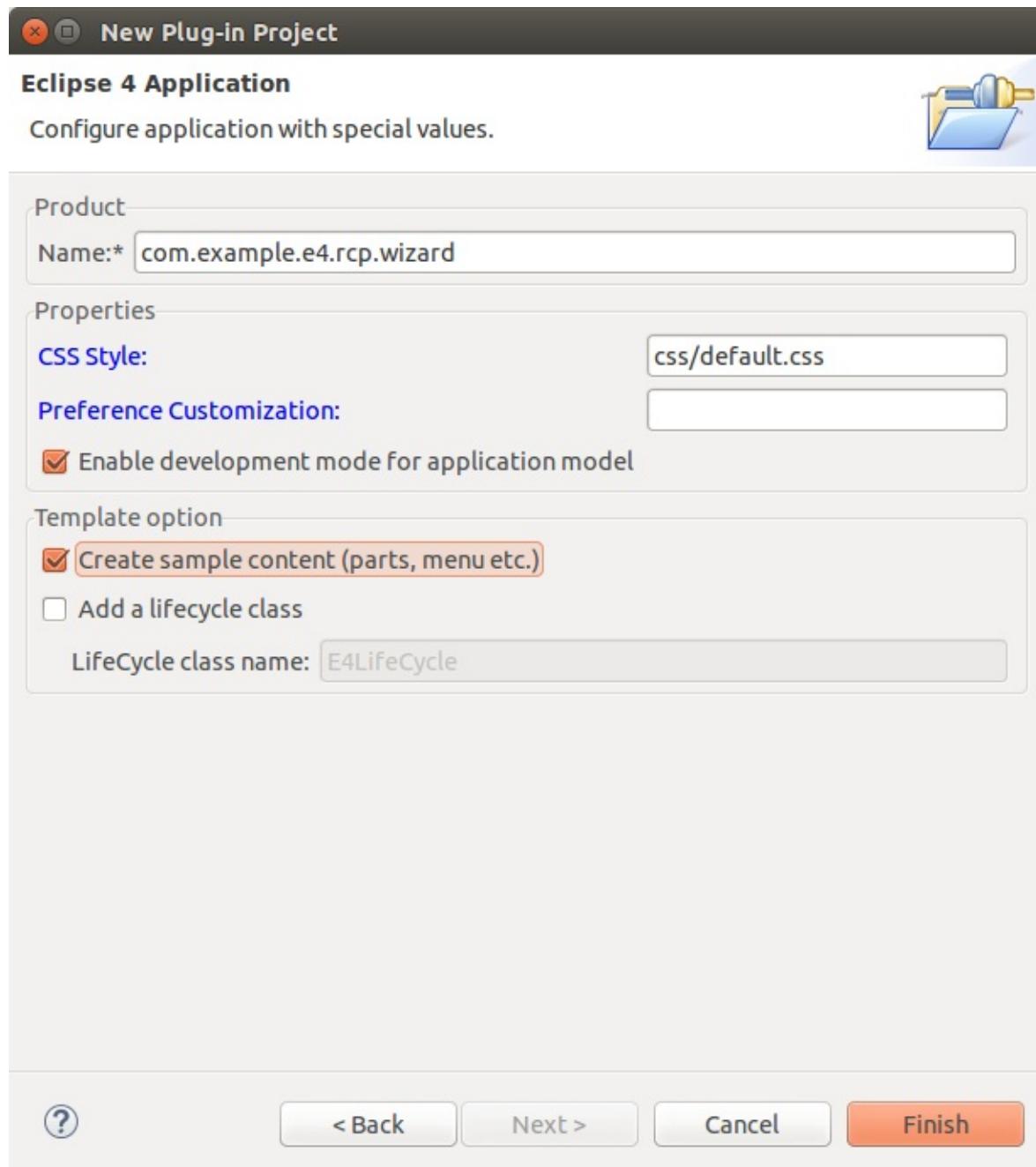


创建一个叫作 com.example.e4.rcp.wizard 的项目，在向导的前 2 页不用修改默认设置。这些设置类似下面的截图：





在向导的第 3 页，勾选 `Enable development mode for application model` 以及 `Create sample content (parts, menu etc.)`。

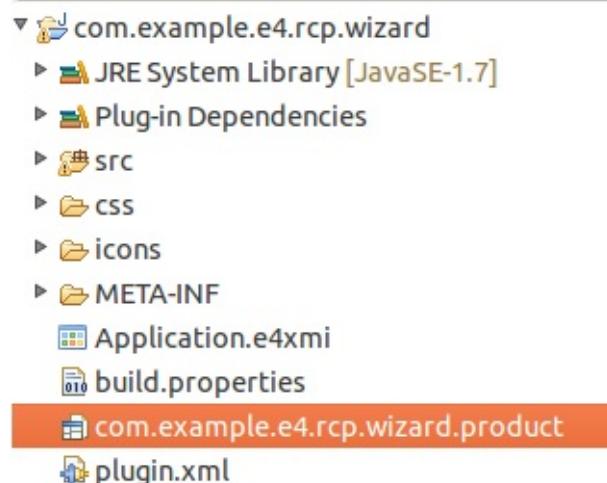


注意： Enable development mode for application model 添加 clearPersistedState 到产品配置文件中，这确保在开发期间，你的应用程序模型总是可见的。参考后面的“在启动时删除持久的用户修改”。通过 Create sample content (parts, menu etc.)，你配置生成的应用程序应该保护例子内容，即一个视图以及一些菜单和工具栏元素。

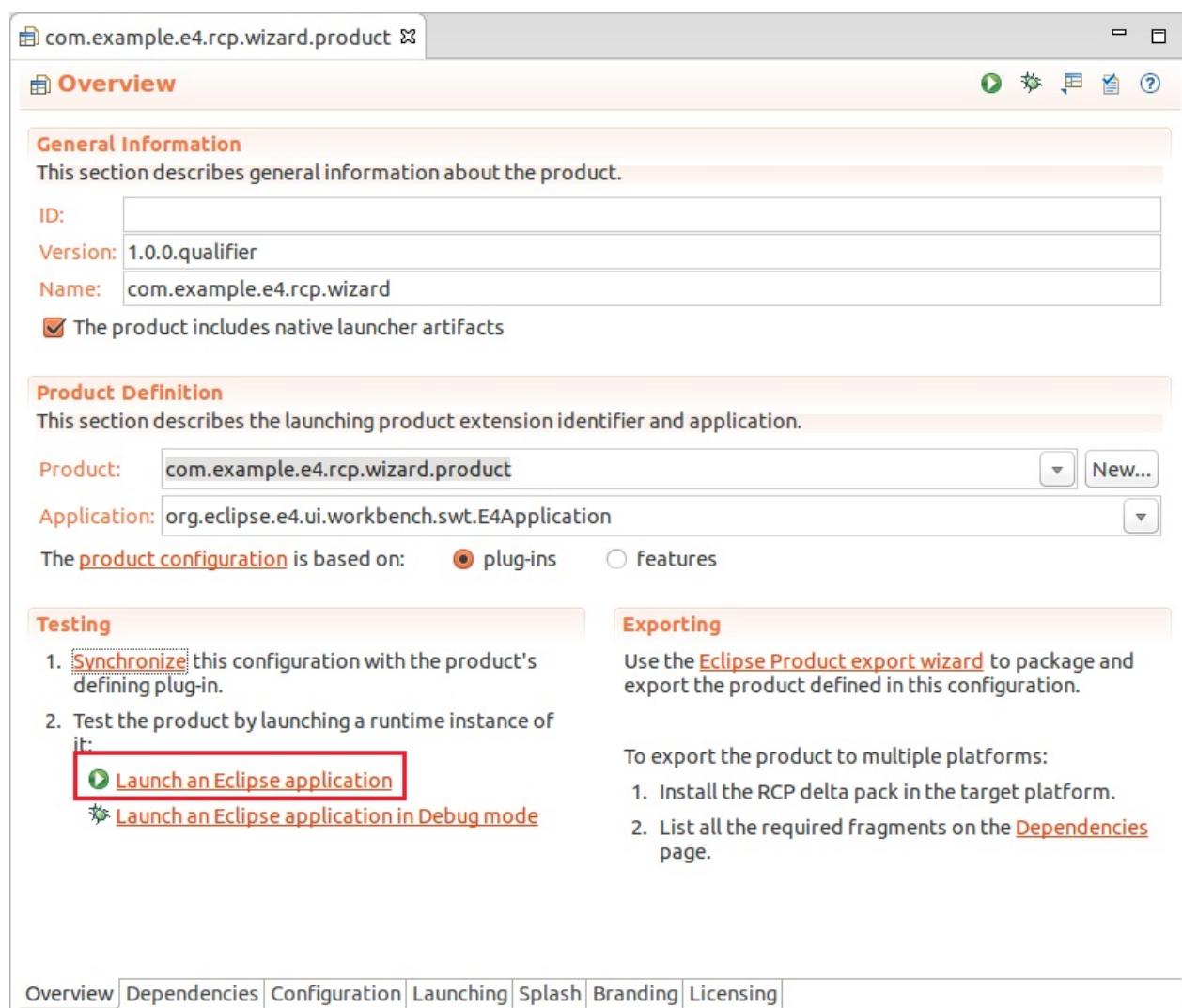
这个向导创建所有需要的文件来启动你的应用程序。

通过产品文件启动你的 Eclipse 应用程序

通过双击打开生成的产品文件。

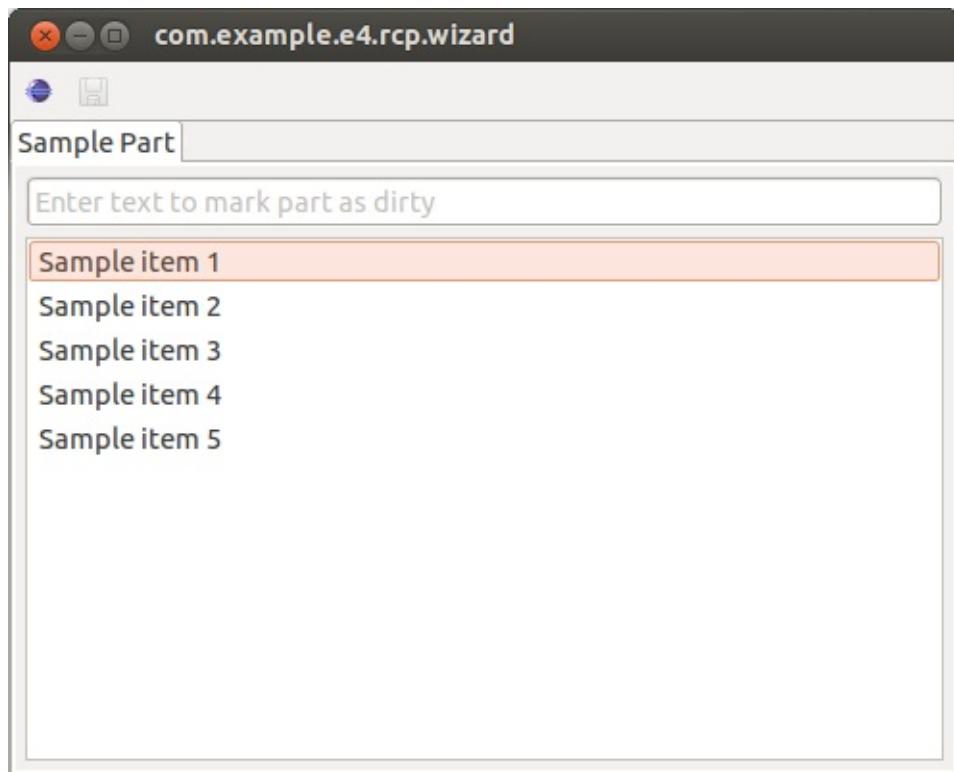


切换到编辑器的 Overview 标签页，然后通过点击 Launch an Eclipse application 超链接启动你的 Eclipse 应用程序。见下图：



验证

最后你的 Eclipse 应用程序应该启动了，应用程序应该看起来类似下面的截图：



使用运行配置

什么是运行配置？

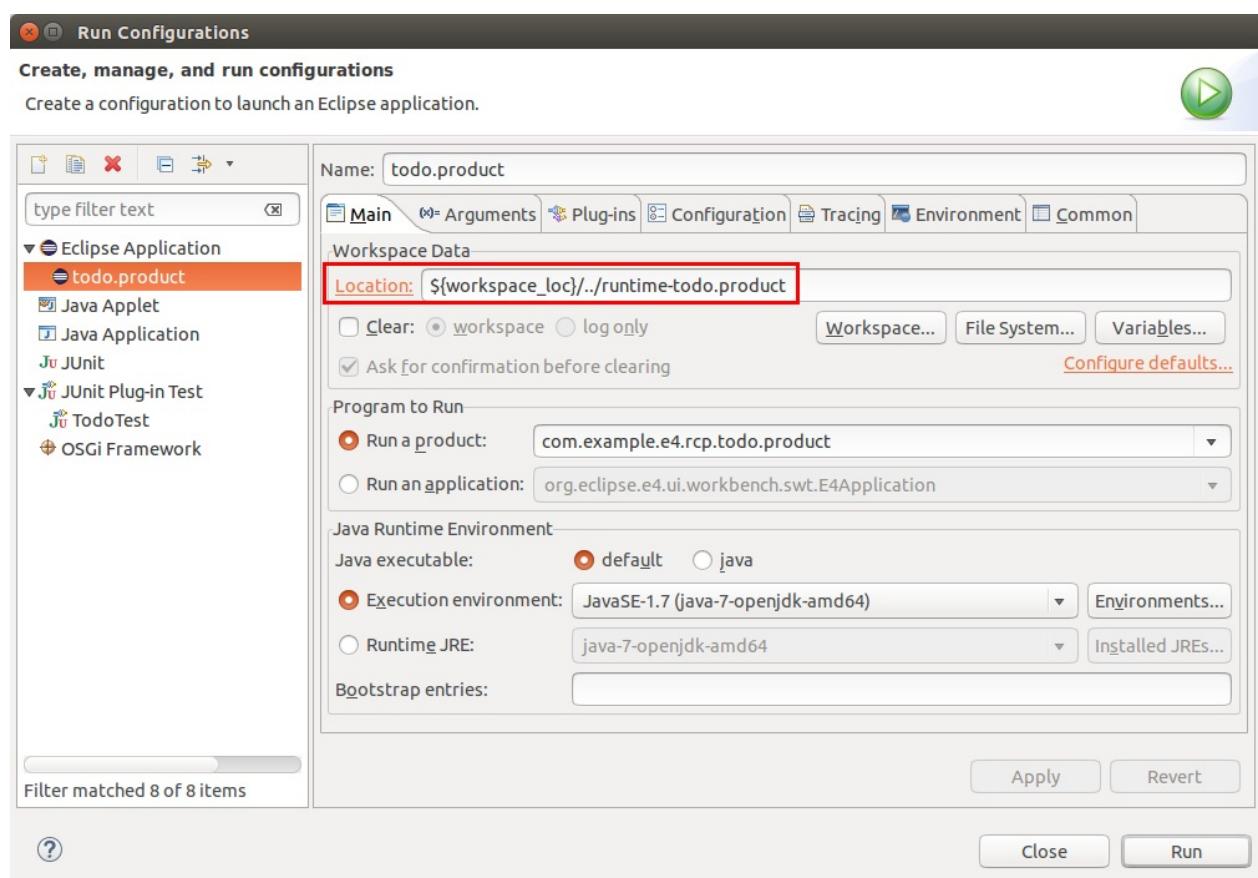
运行时配置定义用来启动的环境，例如，它定义 Java 虚拟机（VM）的参数，插件（classpath）依赖，等等。

如果你启动一个 Eclipse 应用程序，相应的运行配置会自动创建或更新。

检查运行配置

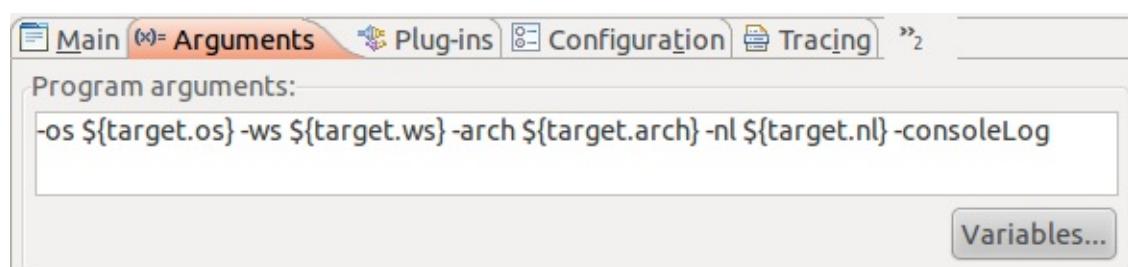
要检查和编辑你的运行配置，从 Eclipse 菜单中选择 **运行 → 运行配置...**。

在主标签页 **Location** 字段指定 Eclipse 将在哪儿创建启动你的 Eclipse 应用程序需要的文件。



运行参数

运行配置允许你在参数标签页中为应用程序添加额外的启动参数。默认情况下 Eclipse 已经包括几个参数，即 -os, -ws 以及 -arch 来指定应用程序运行的体系。



下表列出有用的启动参数。

参数	描述
consoleLog	运行 Eclipse 应用程序的错误消息写到标准输出(System.out), 可以在 Eclipse IDE 控制台查看。
nl	指定应用程序的 locale, locale 定义了语言特殊的设置, 即, 使用什么翻译, 数字, 日期以及货币格式。 例如 -nl en 用英语启动应用程序, 这对于测试翻译是非常有用的。
console	提供到 OSGi 控制台的访问, 你可以检查你的应用程序的状态。
noExit	保持 OSGi 控制台即使在应用程序崩溃的情况下依然打开, 这允许你分析应用程序的依赖关系, 即使应用程序在启动时崩溃了。
clearPersistedState	删除 Eclipse 4 应用程序模型缓存的运行时改变。

启动配置及 Eclipse 产品

启动配置存储产品配置文件的设置，启动配置在你每次通过产品启动应用程序时创建或更新。

你可以直接用创建的运行配置来再次启动应用程序，这种情况下，在产品配置中的改变是不会被考虑的。

警告：更新一个存在的运行配置是常见的失败原因，并且浪费很多时间来分析错误。为了确保你在产品中使用最后的配置，通过产品文件来启动应用程序。

常见的启动问题

Checklist for common launch problems

在 Eclipse RCP 应用程序运行配置中的错误是常见的问题源，本章描述这些问题，在你启动应用程序时有问题，可以参考参考。下表列出了潜在的问题和应对方案。

问题	审查
在启动时显示 "Could not resolve module"	检查在你的产品配置中包含了所有需求的插件。确保你的产品定义了到所有这些需求插件或特征的依赖。参考“在产品启动时查找缺少的插件”。Bundles 可能也需要特定版本的 Java 虚拟机，即，一个 bundle 可能需要 Java 1.6，因此不能在 Java 1.5 VM 中加载。检查 MANIFEST.MF 文件的 Overview 标签页中需求的 Java 版本。
在启动时显示 "java.lang.RuntimeException: No application id has been found."	参考上面的 "Could not resolve module". 大多数情况下也会通过缺少插件依赖触发。
奇怪的行为，但是没有错误消息	检查你的运行配置是否包含 -consoleLog 参数。这个选项允许你在 Eclipse IDE 的控制台视图中查看应用程序错误。
运行时配置经常缺少需求的插件	确保你的产品或特征包含所有需求的依赖。
在产品依赖标签页中的修改没有反映到运行配置（即，添加了新插件但是没有包含在运行配置中）	如果你直接从产品定义文件启动的话，会更新存在的运行配置。如果你直接选择运行配置启动，它将不会更新。
应用程序模型修改没有反映到 Eclipse 4 应用程序中	Eclipse 4 持久化用户的改变到一个文件，在启动时从这个文件中恢复。在开发期间，这可能导致模型修改没有正确的应用到运行时模型的情形，例如，你定义了一个新的菜单项，但是这个菜单项没有在你的应用程序中显示。
服务，例如，key 绑定或选择器服务，在 Eclipse 4 应用程序中不能工作	在 Eclipse 4.3 之前，每个 Part 需要实现一个 @Focus 方法来将焦点放到 SWT 控制中，这个错误在 Eclipse 4.3 之后不再发生。
菜单项在 Eclipse 应用程序中时禁止的	确保包 org.eclipse.e4.ui.internal.workbench.addons 中的 HandlerProcessingAddon 类注册为模型组件。bundle 符号名是 org.eclipse.e4.ui.workbench。
在注册表中找不到应用程序 "org.eclipse.ant.core.antRunner" 或应用程序	确保你在产品配置文件中点击了 新建... 按钮，然后选择了 E4Application 作为启动应用程序。你可以在 plugin.xml 文件中的扩展标签页检查当前的设置，在 org.eclipse.core.runtime.products 扩展细节中。

在产品启动时查找缺少的插件依赖

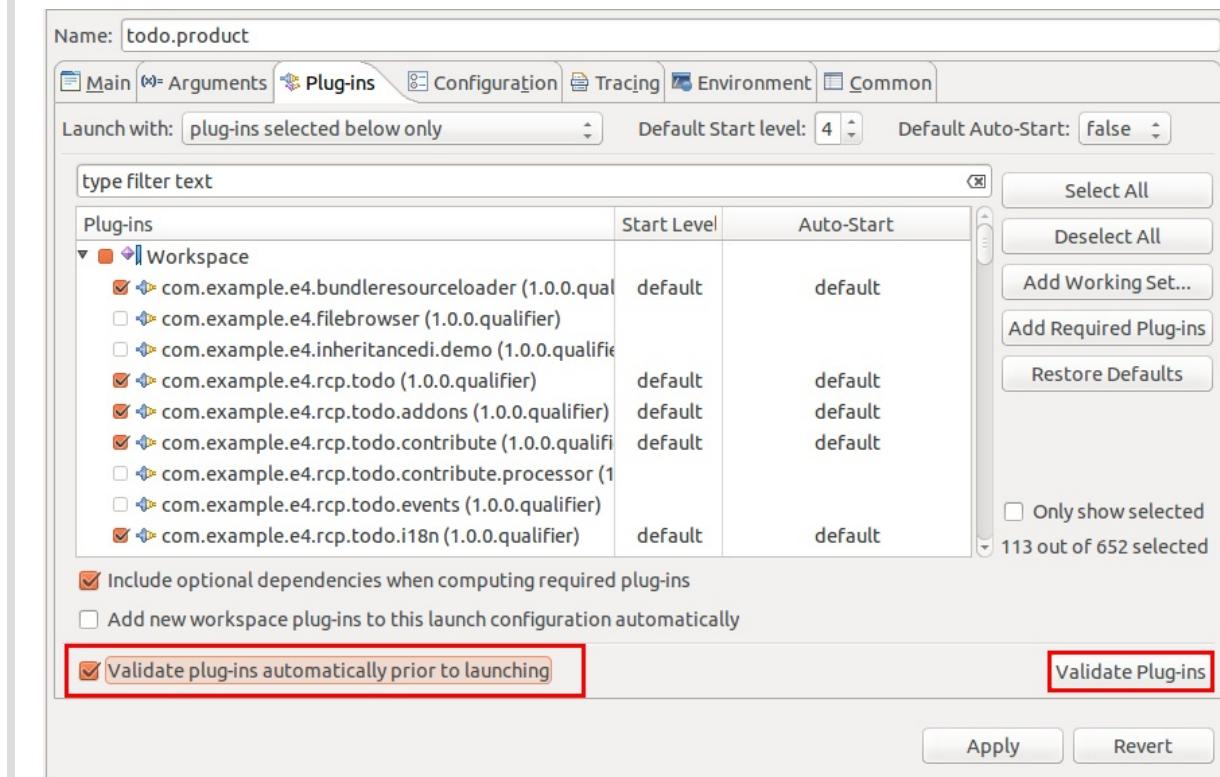
最常见的问题是在你的产品中缺少一些需求的插件，如果你用基于特征的产品配置，你需要确保所有在 MANIFEST.MF 文件中引用的插件也包含在你的特征中。这个错误通过控制台视图报告，通常是第一个错误消息，你需要向上滚动才能看到它。

下面列出了这个消息通常看起来的样子：

```
org.osgi.framework.BundleException:  
  Could not resolve module: com.example.e4.rcp.todo.services [9]  
    Unresolved requirement:  
      Require-Bundle: com.example.e4.rcp.todo.events;  
        bundle-version="1.0.0"
```

在弄清楚缺少的插件后，确保将它们添加到你的产品中（如果产品是基于插件的）或添加到你的特征中（如果产品是基于特征的）。

提示：Eclipse 可以在你运行启动配置之前自动的检查缺少的依赖。在插件标签页，点击 验证插件 按钮或选择 在启动之前自动验证插件 选项。这将检查运行配置是否有所有需求的插件。



避免在运行配置中修复依赖问题，因为运行配置是基于产品配置文件创建或更新的，因此总是确保产品文件是正确配置的，而不是修改派生的信息。产品配置用来输出你的产品，因此产品依赖中的错误会导致输出的应用程序不能启动。

Eclipse 4 应用程序模型

什么是应用程序模型？

Eclipse 平台用一个称为 `应用程序模型` 的抽象来描述一个应用程序的结构，`应用程序模型` 包含可见元素，也包含不可见元素。可见元素包括窗口、Parts(视图和编辑器)，菜单、工具栏等等。不可见元素包括 Handlers、Commands、按键绑定等等。

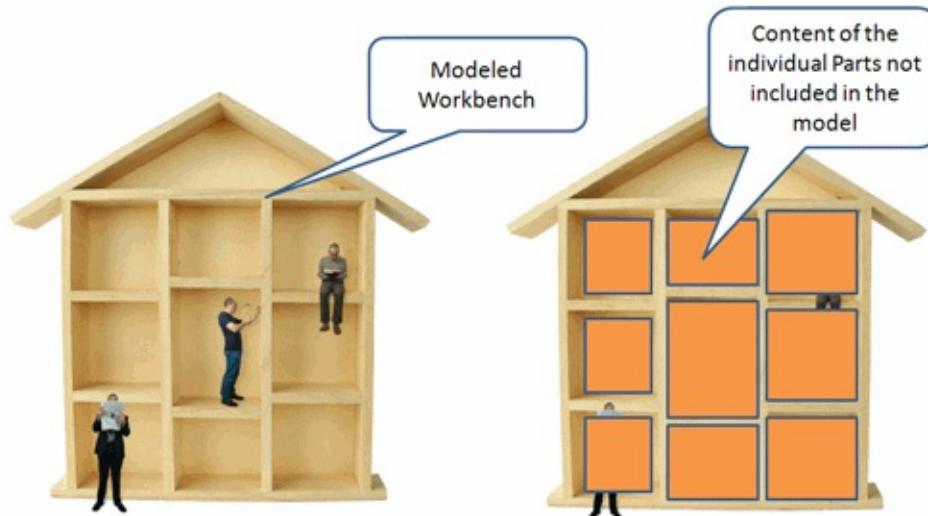
每个模型元素包含属性来描述它当前的状态，例如窗口的位置和大小，`应用程序模型` 也通过层次来表达模型元素之间的关系。

应用程序模型范围

`应用程序模型` 定义了应用程序的结构，例如，它描述了哪些 Parts 是有效的，也描述了 Parts 的属性，例如，Part 是否可以关闭，它的标签，ID，等等。

在 Part 中显示的独立用户接口控件，不是通过 `应用程序模型` 定义的，即，Part 的内容还是通过源代码来定义。

如果 `应用程序模型` 是一间屋子，它将描述有效的房间（Parts）以及它们的排列方式（透视图，Part 栈，Part Sash 容器），但是不包括房间里的家具，这通过下图解释。



你如何定义应用程序模型？

`应用程序模型` 的基础通常定义为静态文件，默认的，它叫作 `Application.e4xmi`，放在插件的主目录中，定义了产品的扩展。

这个文件在应用程序启动时读取，用来构造最初的应用程序模型，用户的修改被持久化保存，并在启动时重新应用。

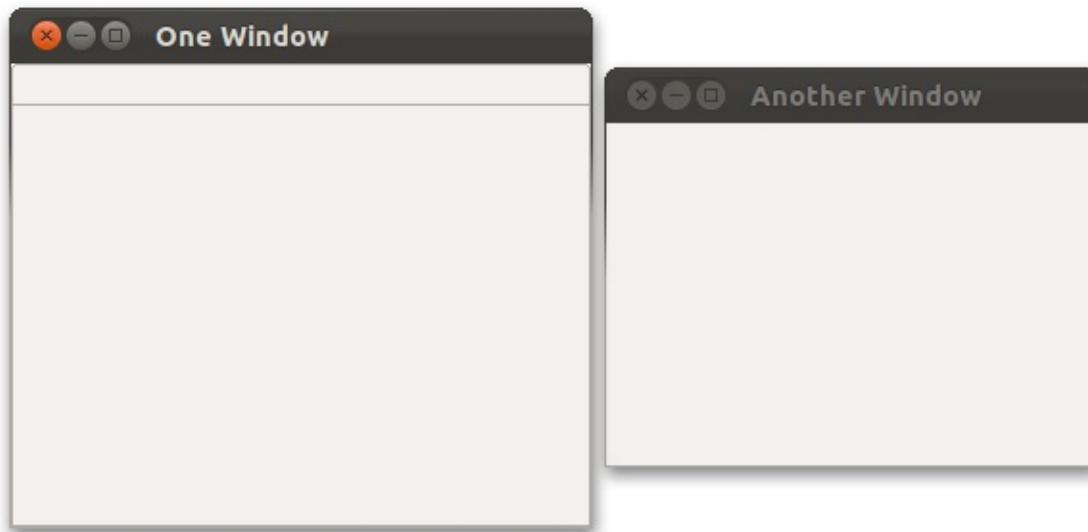
`应用程序模型` 是可扩展的，例如，其它插件可以通过 `模型处理器` (Model Processors) 以及 `模型碎片` (Model Fragment) 来为它贡献内容。

用户接口模型元素

下面的模型元素表示你可以用来创建用户应用程序接口的基本元素。

窗口

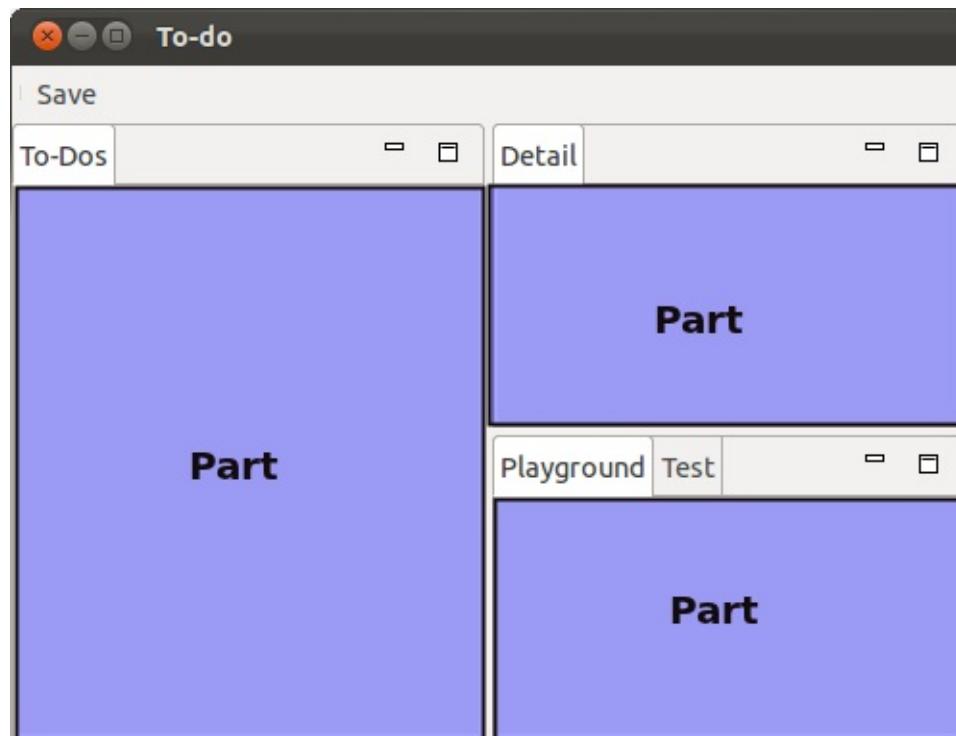
应用程序由一个或多个窗口组成，通常一个应用程序只有一个窗口，但并不限制这样，例如你连接了多个显示器，可以打开多个窗口。



Parts

什么是 parts?

Parts 是你用来导航或修改数据的 UI 元素，一个 Part 可有一个下拉菜单，一个上下文菜单，以及一个工具栏。Part 可以堆叠，或者放到其他 Part 后面，这依赖于它们放入的容器。



Parts behaving as views or as editors

Part 通常分为 视图 和 编辑器，区分它们并不是因为技术上的不同，而是使用它们的概念不同。

视图通常在一组数据上工作，可能是一个层次数据结构，如果数据通过视图改变了，这个改变通常直接反映到底层的数据结构，视图有时允许用户针对选择的数据打开编辑器。

在 Eclipse IDE 中视图的一个例子是包浏览器，允许你浏览项目文件，如果你改变包浏览器中的数据，例如，你重命名文件，文件名会直接在文件系统中改变。

编辑器通常用来修改单个数据元素，例如文件内容或对象数据，要将编辑器中的修改反映到底层数据结构，用户需要明确的保存编辑器内容。

例如，Java 编辑器用来编辑 Java 源文件，改变源文件需要选择 保存 按钮。一个未保存的编辑器标签页标题在文件名左边有一个星号。



Part 容器

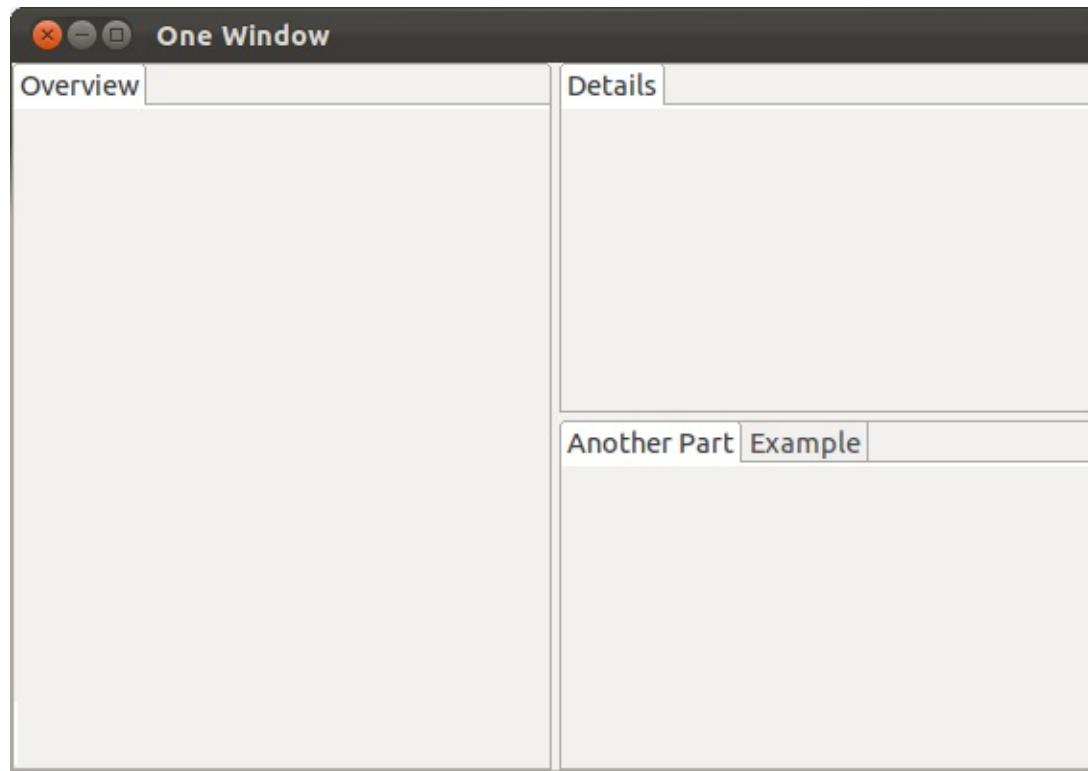
Part stack 及 part sash 容器

Part 可以直接附属到窗口或透视图，它们也可以通过附加的模型元素组织排列，例如通过 Part 栈或者通过 Part Sash 容器元素。

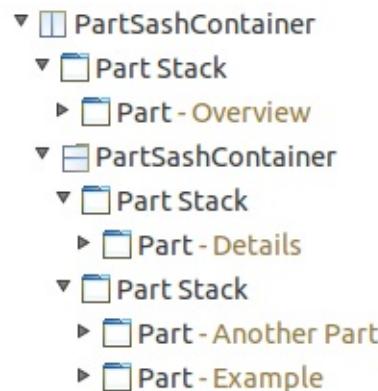
一个 Part 栈包含一组 Parts，一次只显示一个 Part 的内容，但是显示所有 Part 的标题标签。用户可以通过选择对应的标题标签来激活某个 Part。

Part Sash 容器同时显示所有的子元素，或者垂直的分布，或者水平的分布。

下面的屏幕截图显示了用 2 个 Part Sash 容器以及 3 个 Part 栈的 Eclipse 应用程序布局。

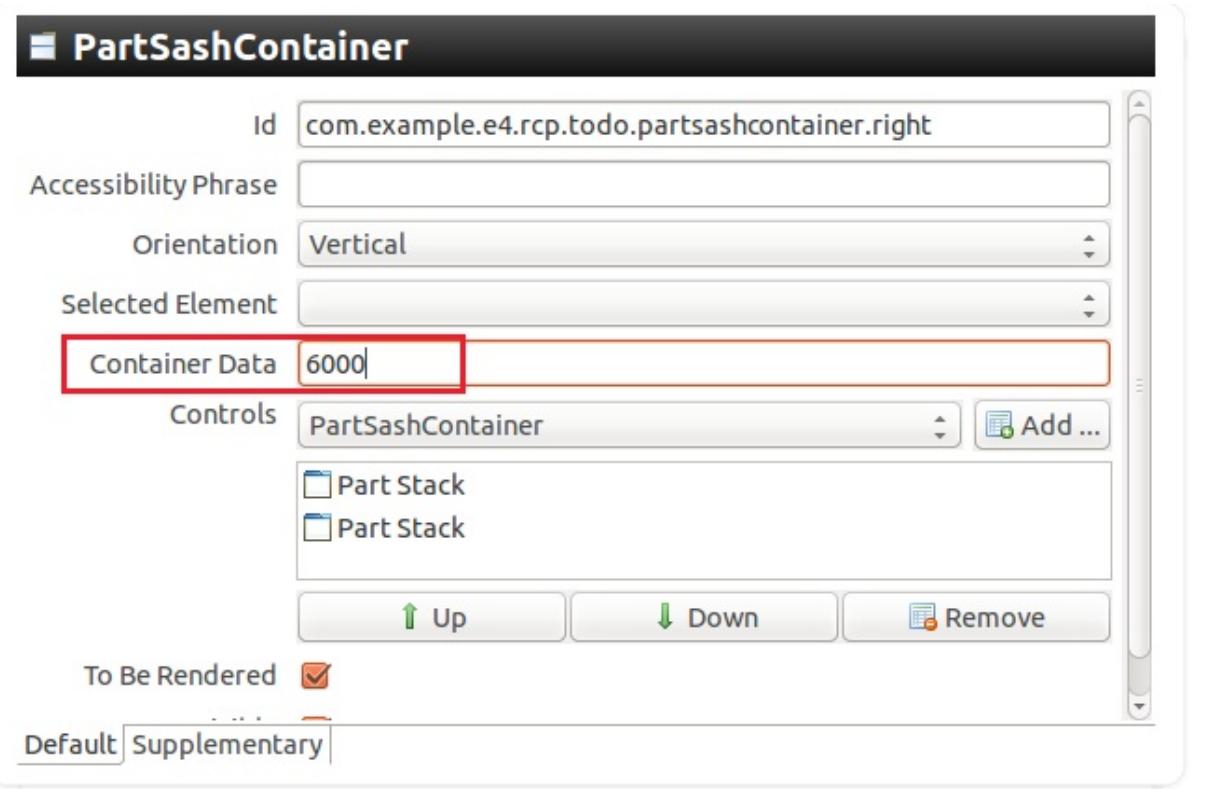


在上面的布局中，有一个水平的 Sash 容器包含另一个 Sash 容器以及一个 Part 栈，二级 Sash 容器包含 2 个 Part 栈，这个层次可以用下面的图描述：



为子元素使用布局权重数据

你可以用 Part Sash 容器的子元素的 容器数据 来分配布局权重。布局权重解释为相对于其他子元素的空间需求，设置参考下图：



如果你为容器的一个子元素设置了 容器数据，那么你必须为所有其它子元素也定义 容器数据，否则缺少定义的子元素将得到一个非常高的值，并占用所有有效的空间。

提示，容器所有的总空间初始值是在子元素移动的时候维护，为了平滑，这个总值必须接近屏幕精度，太小的值(例如 50/50)导致每个框格单元移动多个像素，用户感觉是不是自己移动的太快了。

透视图

透视图是可选的容器，包括一组 Parts，透视图可以用来存储不同的 Parts 布局，例如 Eclipse IDE 用它们来布局对应到开发人员希望执行的任务（开发、调试，等等）。

你可以将透视图放在应用程序模型的透视图栈中，可以通过 Eclipse 平台提供的 Parts 服务来切换透视图。

连接模型元素到类和资源

URI 模式或类和资源

模型元素可以通过统一资源标识(URI)指向一个类或者静态资源。Eclipse 定义了 2 个 URI 模式用作这种通途，下面描述了这些模式，例子中假设 bundle 叫做 test。

bundleclass://BSN/package.classname

例子： `bundleclass://test/test.parts.MySavePart`

一个 Java 类标识

它有下列部分组成：**bundleclass://** 是固定前缀，BSN 表示 Bundle-SymbolicName，在 MANIFEST.MF 配置文件中定义，Bundle-SymbolicName 后跟一个 / 以及全资格类名。

platform:/plugin/BSN/path/filename.extension

例子： `platform:/plugin/test/icons/save_edit.gif`

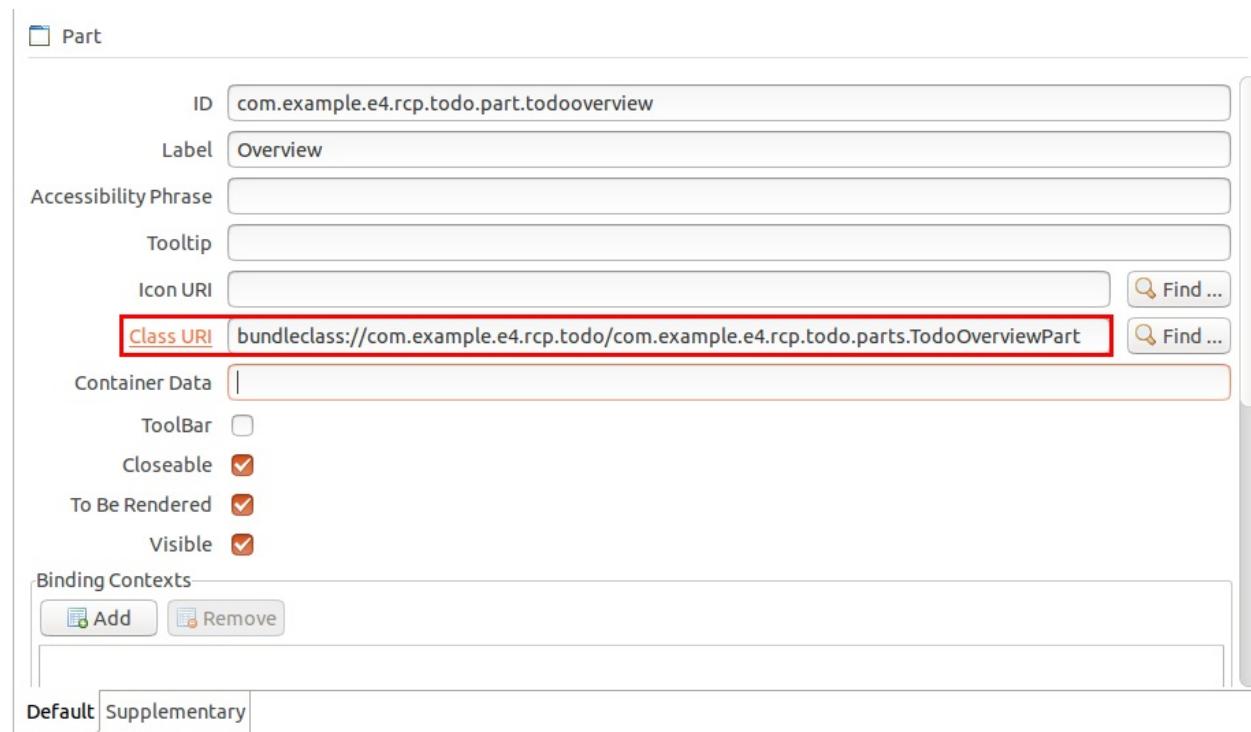
用来标识资源

标识插件中的一个资源，**platform:/plugin/**是固定前缀，BSN 表示 Bundle-SymbolicName，在 MANIFEST.MF 配置文件中定义，Bundle-SymbolicName 后跟文件的路径和文件名。

连接模型元素到类

几个应用程序模型元素，例如 Part，有一个 Class URI 属性，通过 `bundleclass://` URI 指向一个 Java 类。这个类提供了 Part 的行为，相应的对象由 Eclipse 框架创建。

用前面的 房子/房间 作比喻，类 负责 定义房间内的布局，家具等等。



Eclipse 延迟实例化引用的类对象（对于大部分模型元素），这意味着，例如，Part 的类是在 Part 变的可见时才实例化。

连接模型到资源

几个模型元素可以用 `platform:/plugin/` URI 指向静态资源，Part 模型元素包含 Icon URI 属性，可以用来指向在这个 part 中使用的图标。

模型对象以及运行时应用程序模型

模型对象

在启动时， Eclipse 平台解析关于应用程序模型有效的信息（Application.e4xmi，持久化用户修改，以及模型贡献），然后通过 Java 对象存储这些信息，这些对象称为模型对象，在运行时它们表示模型元素的属性。

下表列出了重要的模型对象类型。

模型元素	描述
MApplication	描述应用程序对象，所有其他模型元素是包含在这个对象中。
MAddon	一个自包含的组件，通常不需要用户交互。它可以在应用程序生命周期内注册事件和处理这些事件。
MWindow	表示一个应用程序窗口
MTrimmedWindow	类似于 MWindow，但是它允许包含窗口工具栏（通过 TrimBars 模型元素）。
MPerspective	表示一个在窗口中显示不同的布局，应该被包含在 MPerspectiveStack 中。
MPart	表示一个模型元素 part，例如一个视图或一个编辑器。
MDirtyable	MPart 的属性，可以被注入，如果设置为 true，这个属性告诉 Eclipse 平台这个 Part 包含未保存的数据。在处理器中你可以查询这个属性来提供可能的保存。
MPartDescriptor	是新 Parts 的模板，一个基于这个 part 描述的新 part 可以通过 Eclipse 平台来创建和显示。
Snippets	Snippets 可以用来预配置你希望通过程序建立的模型部分，你可以用 Eclipse 框架来克隆 snippet，然后将结果对象附属到应用程序。

运行时应用程序模型

建立的一系列模型对象通常称为运行时对象模型，运行时应用程序模型是动态的，也就是说，你可以改变这些模型以及它们的属性，并且这些改变会在应用程序中得到反映，Eclipse 平台在模型对象上注册了事件监听器，当你改变相关属性时，会更新用户界面。

特征及产品

The following description uses feature projects and products, please see [Feature projects](#) and [Eclipse Products and Deployment](#) for a description of these topics.

练习：建立一个插件

目标

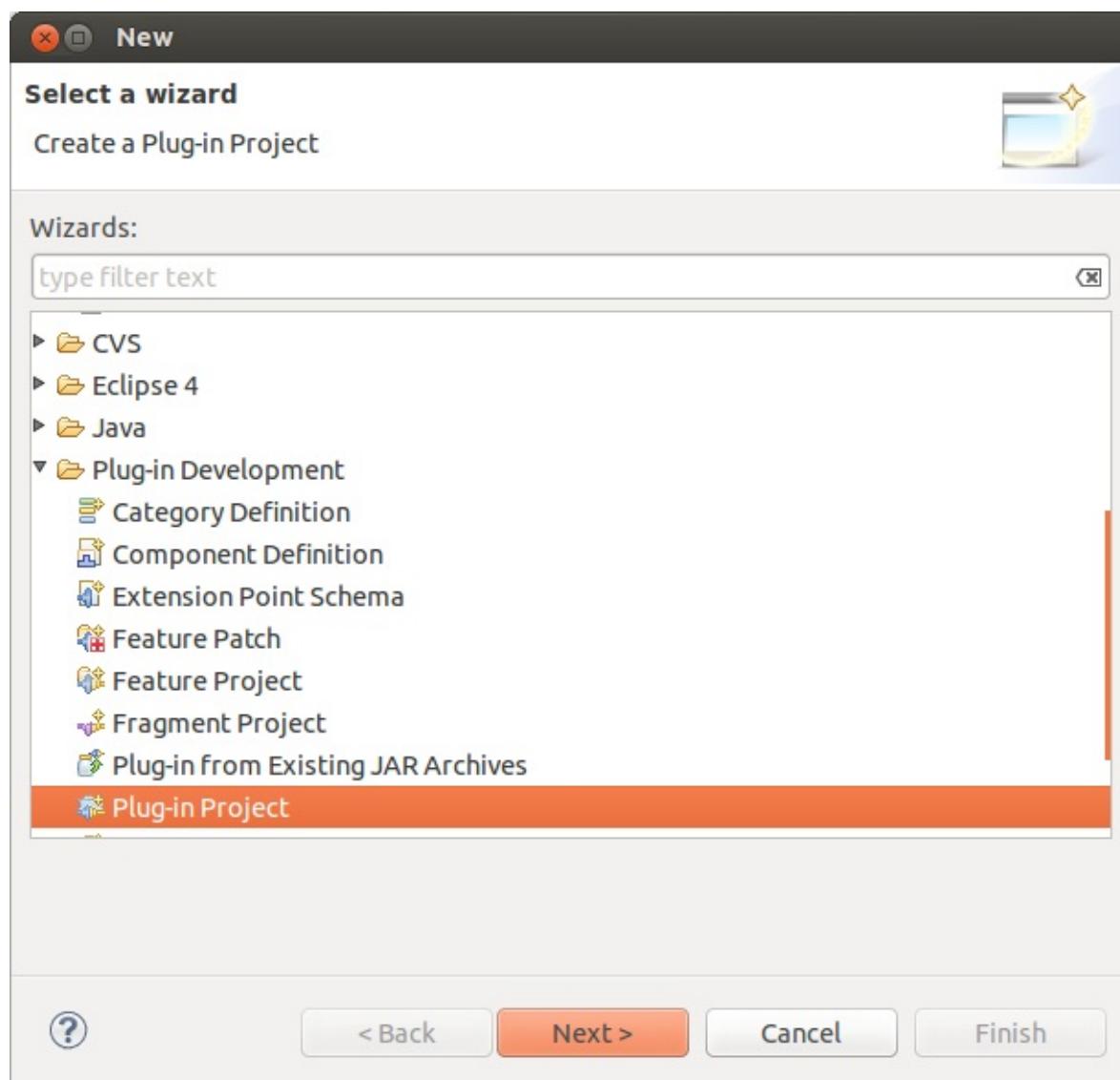
在本练习中，你将建立一个插件，后面我们会将这个插件转换为 Eclipse RCP 应用程序。

注意：后面的描述将此插件称为 **应用程序插件**，这个插件将包含主要的应用程序逻辑。

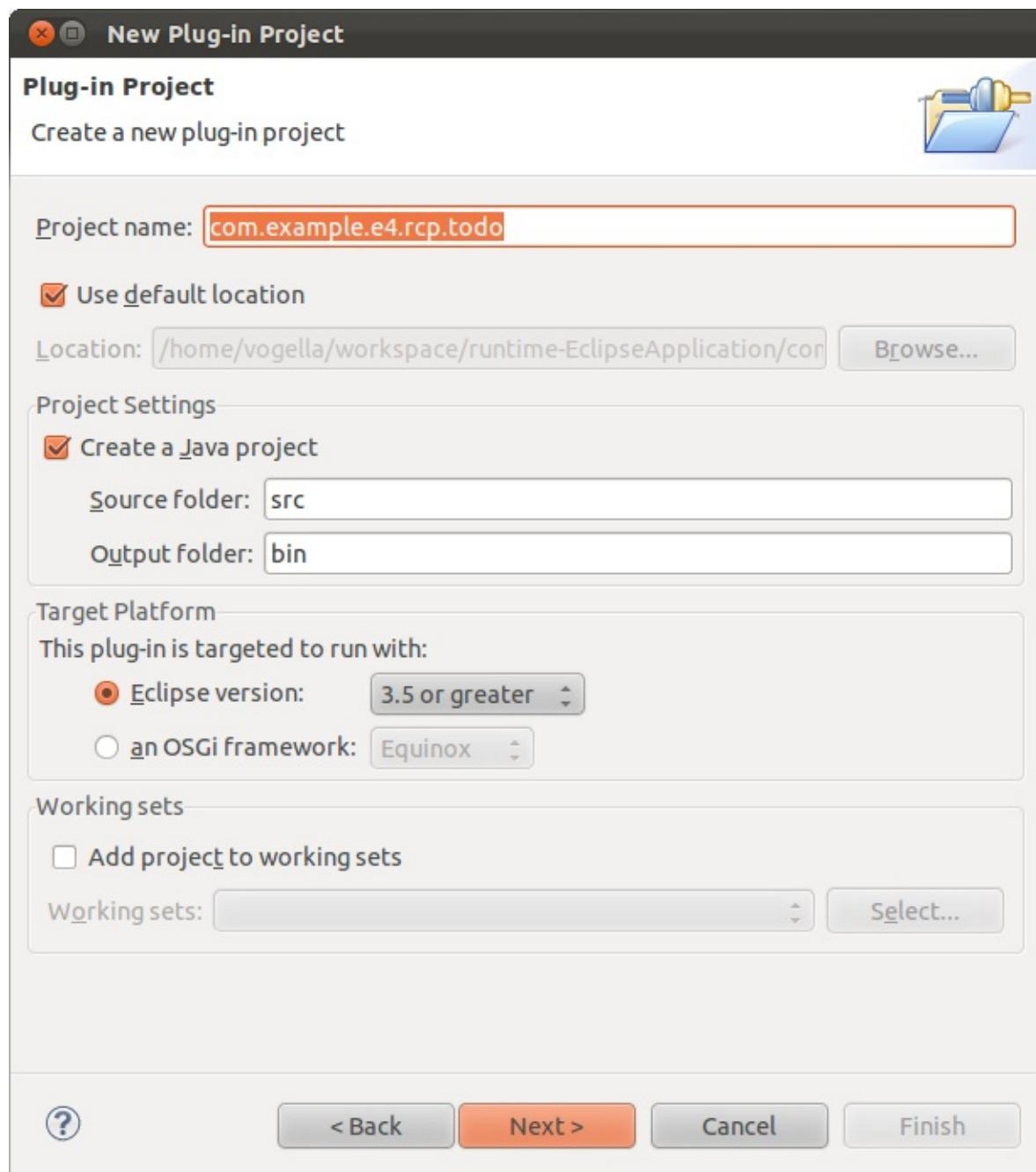
注意：Eclipse 4.5 或以上版本显示和截图略有区别，请在练习时注意正确区分。

建立一个插件项目

在 Eclipse 中选择 **文件** → **新建** → **其它...** → **插件开发** → **插件项目**。

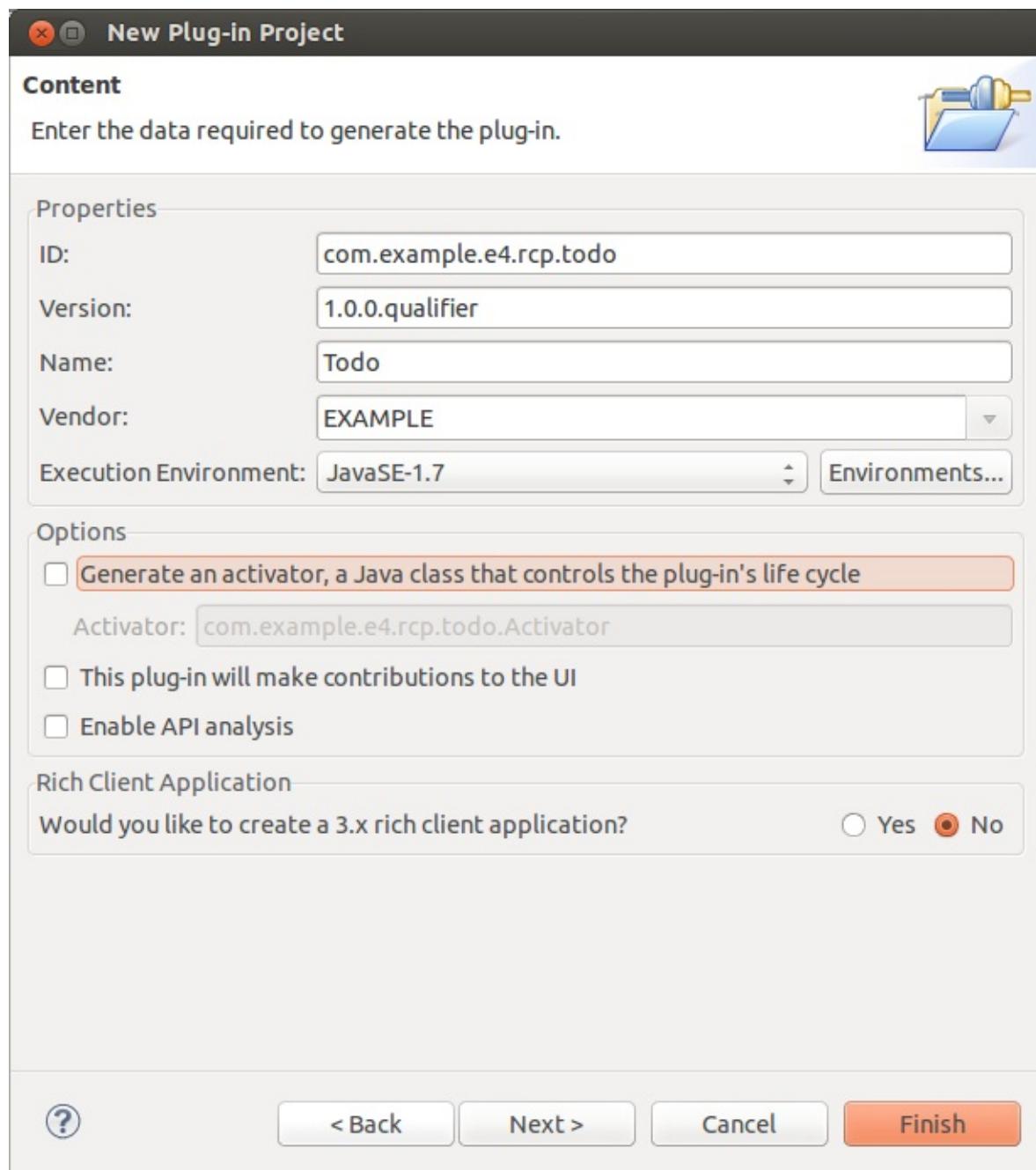


将插件命名为 `com.example.e4.rcp.todo`，然后选择 **下一步** 按钮。



在下一步向导页中，设置如下：

Would you like to create a 3.x rich client application? 选项选择 No ;
 不要勾选 This plug-in will make contributions to the UI ;
 不要勾选 Generate an activator, a Java class that controls the plug-in's life cycle 。



警告：Would you like to create a 3.x rich client application? 以及 This plug-in will make contributions to the UI 选项是与 Eclipse 3.x API 兼容应用程序相关的，如果你希望使用 Eclipse 4 API，永远不要用这些选项。

注意：在 Eclipse 4.5 中，这个向导页被重新设计了，也可以通过它建立一个 Eclipse 4 RCP 应用程序，选择 ...create rich client ... 标志，然后在下一页中选择 Eclipse 4 模板。

点击 完成 按钮，如果你点击 下一步 按钮而不是 完成，向导将显示一个可以跳过的模板选择页。

验证结果

打开项目，确保没有在源代码文件夹中建立 Java 类。

在 manifest 编辑器中，切换到 Dependencies 标签，确保这里没有任何依赖项。

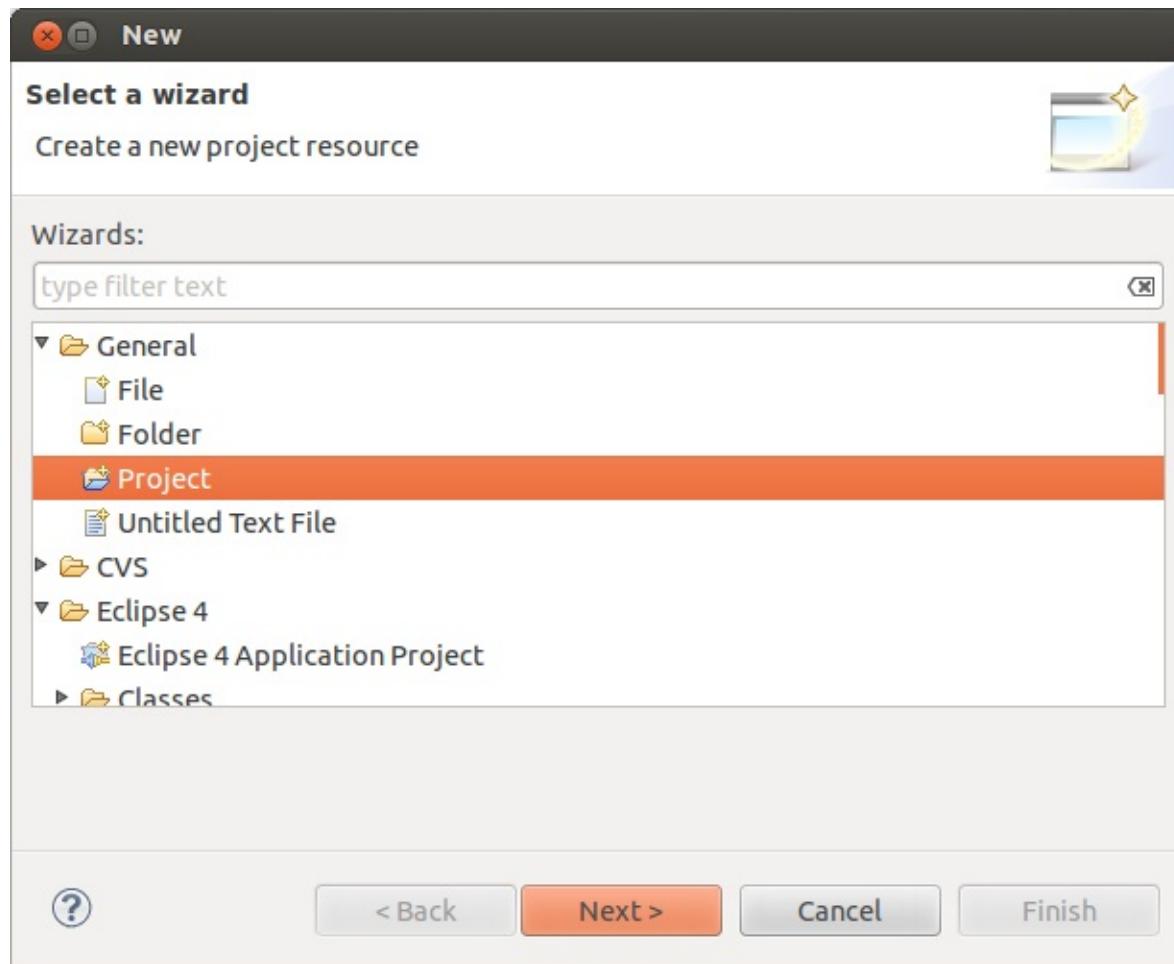
练习：转换插件为 Eclipse 4 应用程序

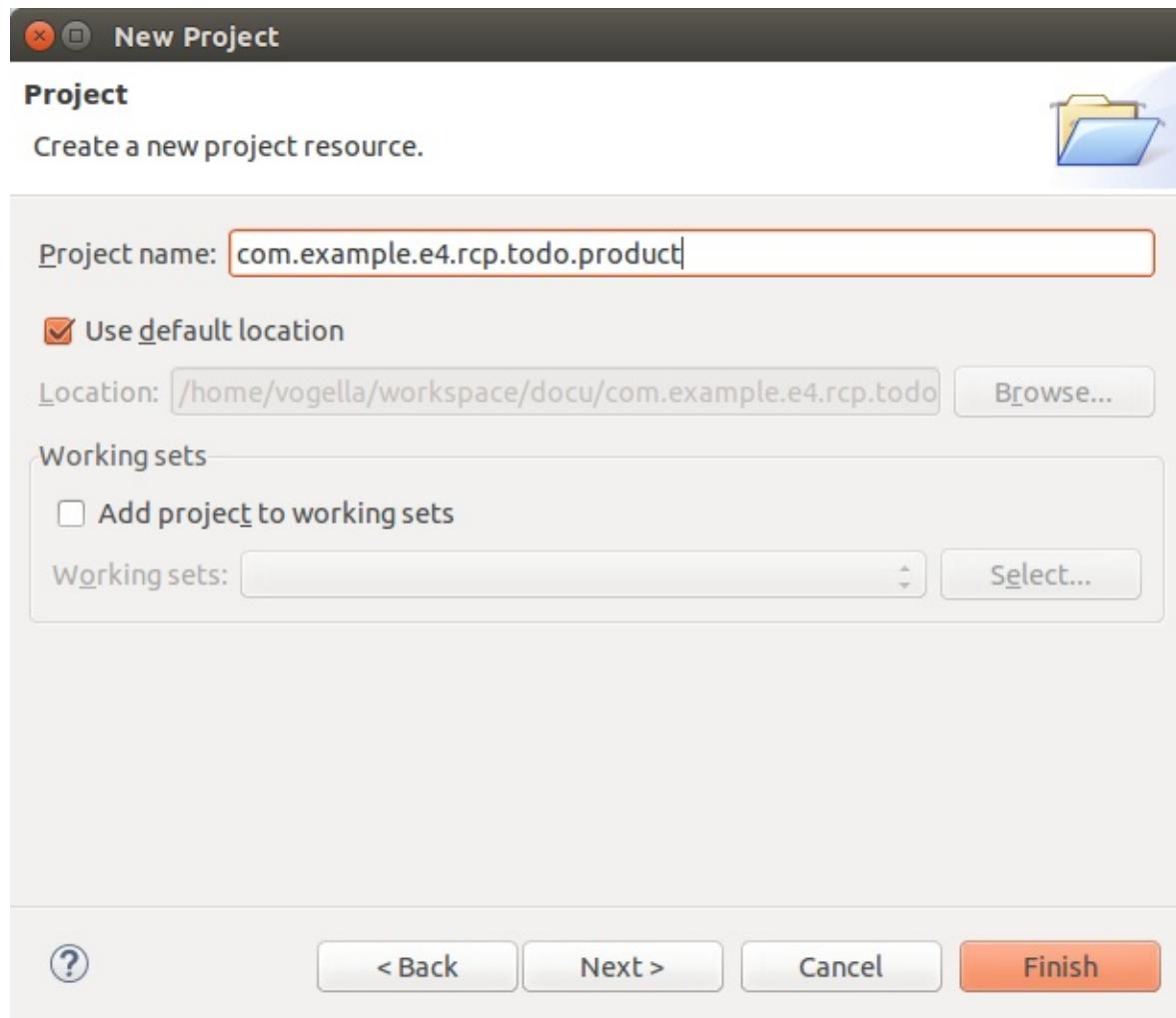
目标

在本章中，你将转换前面生成的插件到一个 Eclipse 4 应用程序。

建立一个项目来作为产品配置文件的宿主

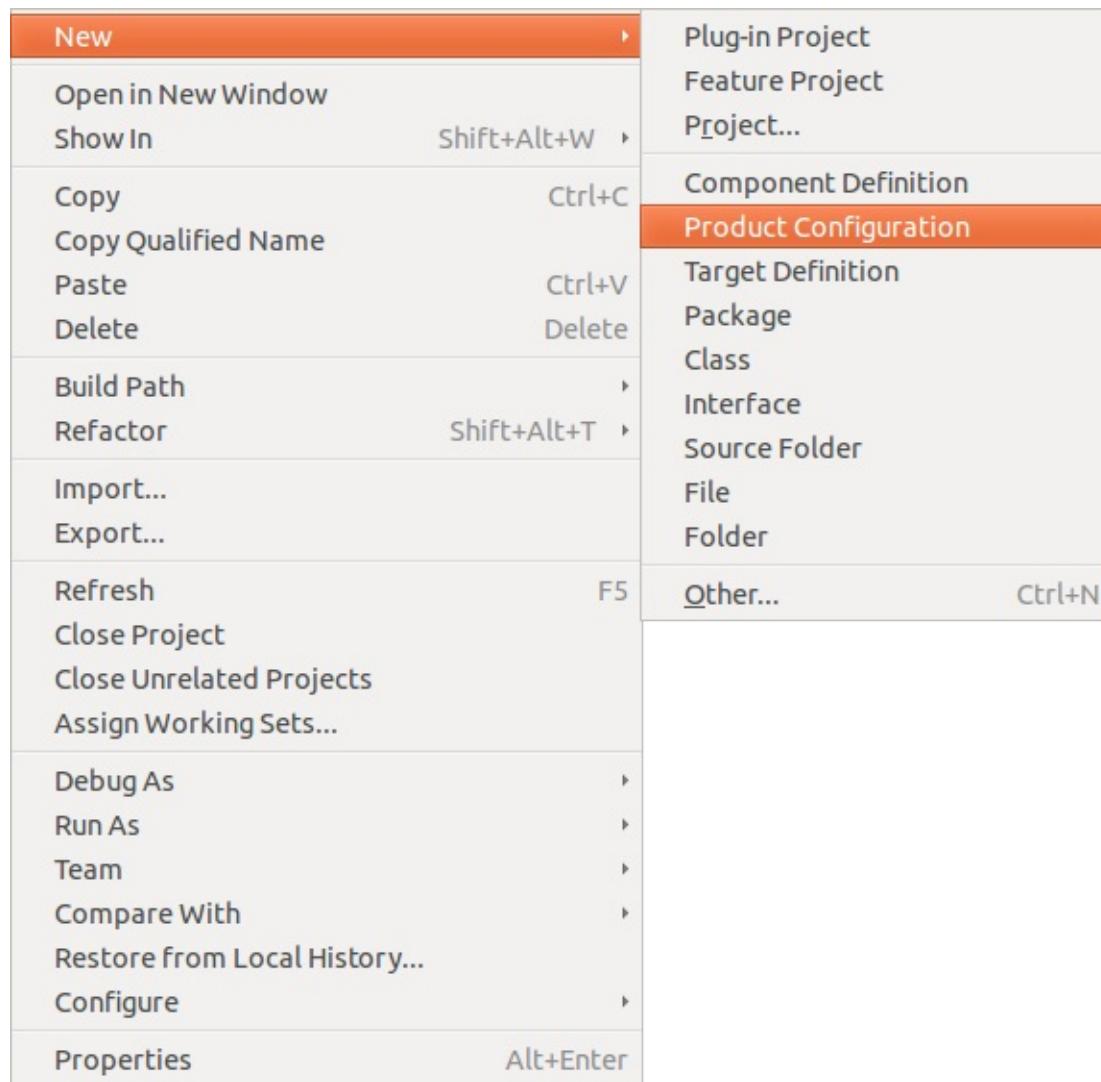
通过 文件 → 新建 → 其它... → 普通 → 项目 建立一个叫做 com.example.e4.rcp.todo.product 的项目。



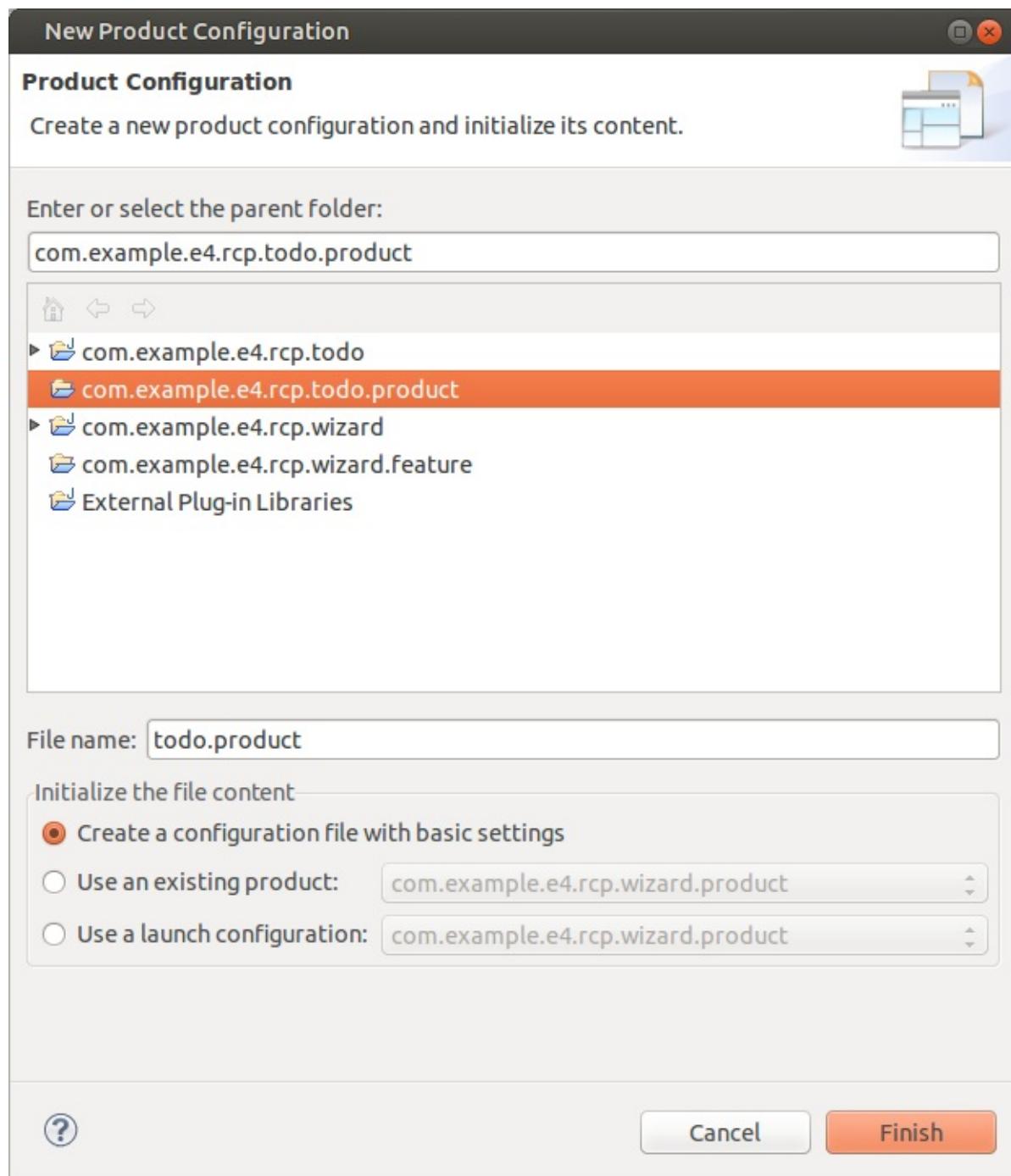


建立一个产品配置文件

在项目上右击，然后选择 新建 → 产品配置。



在 `com.example.e4.rcp.todo.product` 文件夹中建立一个叫做 `todo.product` 的产品配置文件。



点击 完成 按钮，这个文件就建立了，并且在编辑器中打开。

在产品编辑器的 overview 标签页中选择 新建...。

General Information
This section describes general information about the product.

ID:

Version:

Name:

The product includes native launcher artifacts

Product Definition
This section describes the launching product extension identifier and application.

Product:

Application:

The [product configuration](#) is based on: plug-ins features

Testing

1. [Synchronize](#) this configuration with the product's defining plug-in.
2. Test the product by launching a runtime instance of it:

[Launch an Eclipse application](#)

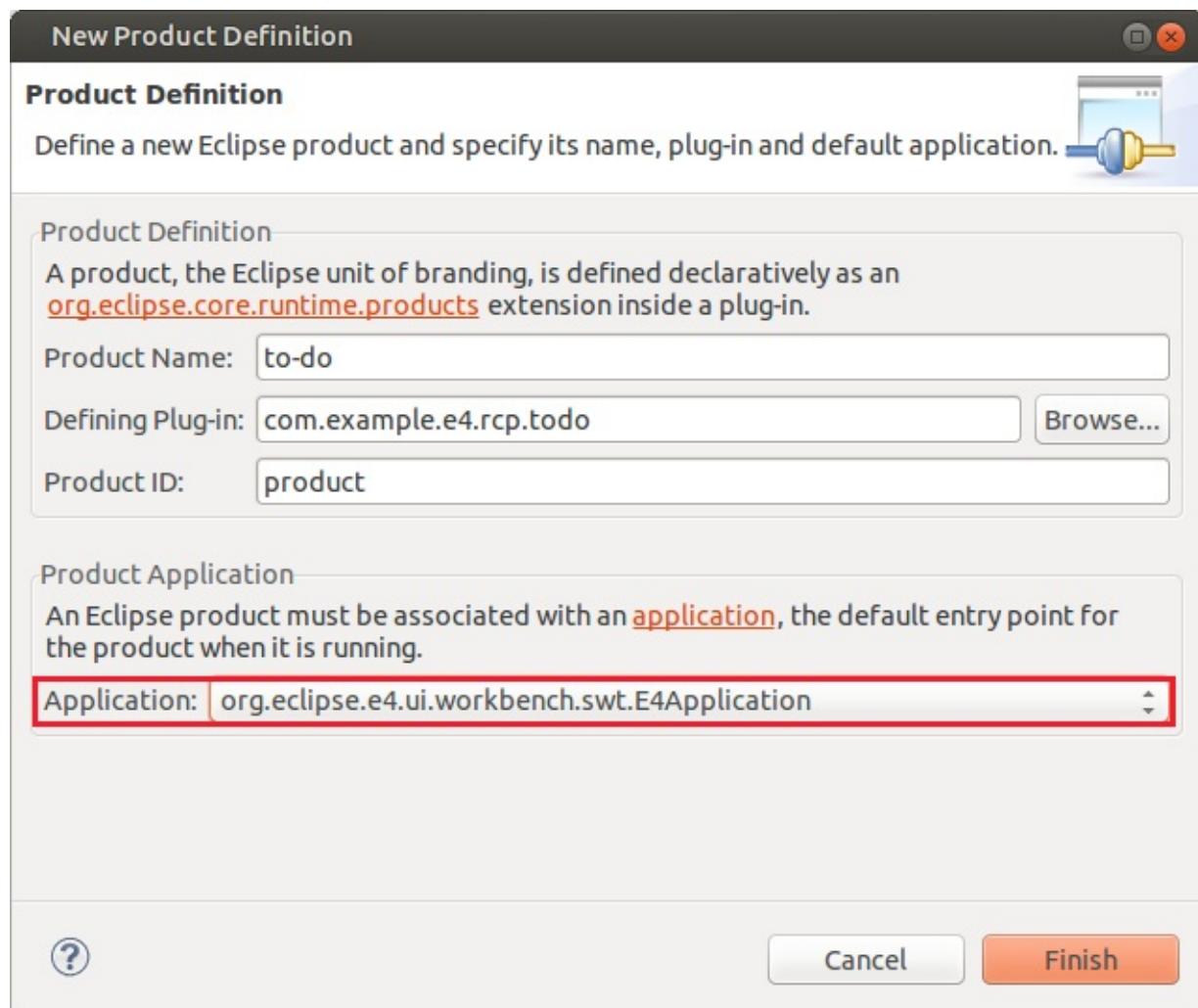
Exporting

Use the [Eclipse Product export wizard](#) to package and export the product defined in this configuration.

To export the product to multiple platforms:

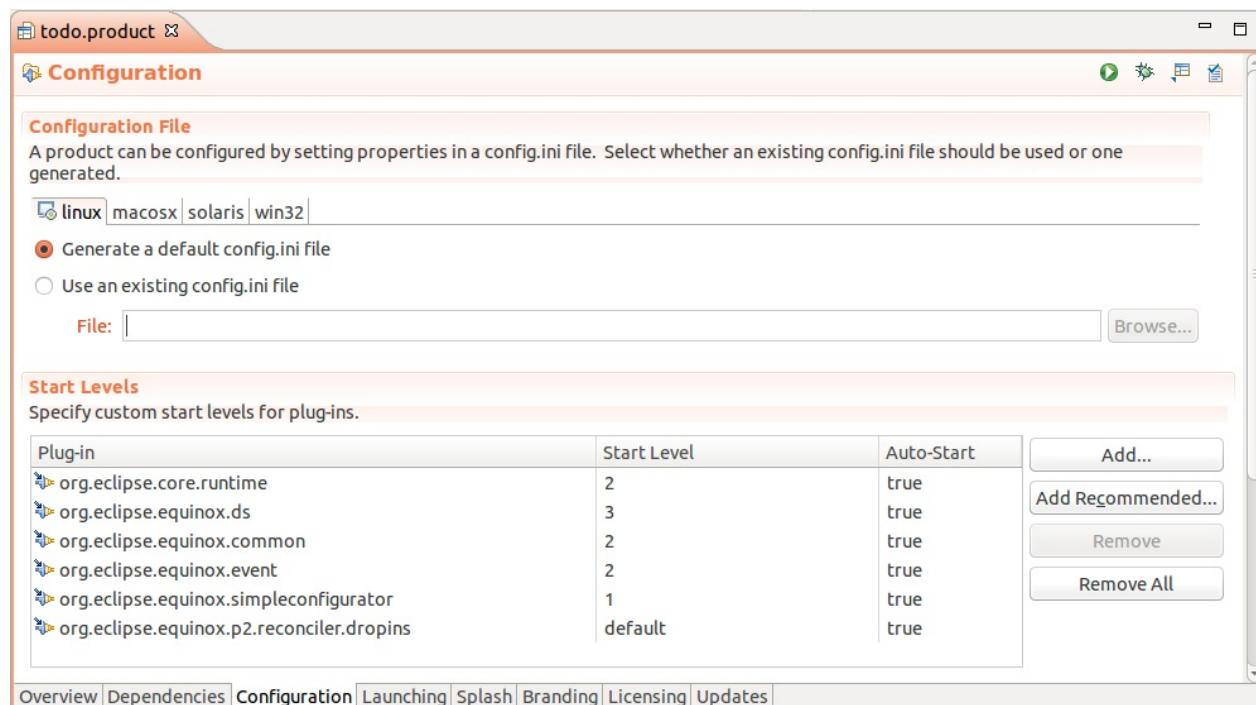
[Overview](#) | [Dependencies](#) | [Configuration](#) | [Launching](#) | [Splash](#) | [Branding](#) | [Licensing](#)

输入 `to-do` 作为产品名称，你的插件作为定义的插件，`product` 作为产品ID，然后在应用程序组合框中选择 `org.eclipse.e4.ui.workbench.swt.E4Application`。



配置启动级别

在产品编辑器中切换到配置标签页，点击 Add Recommended... 按钮。

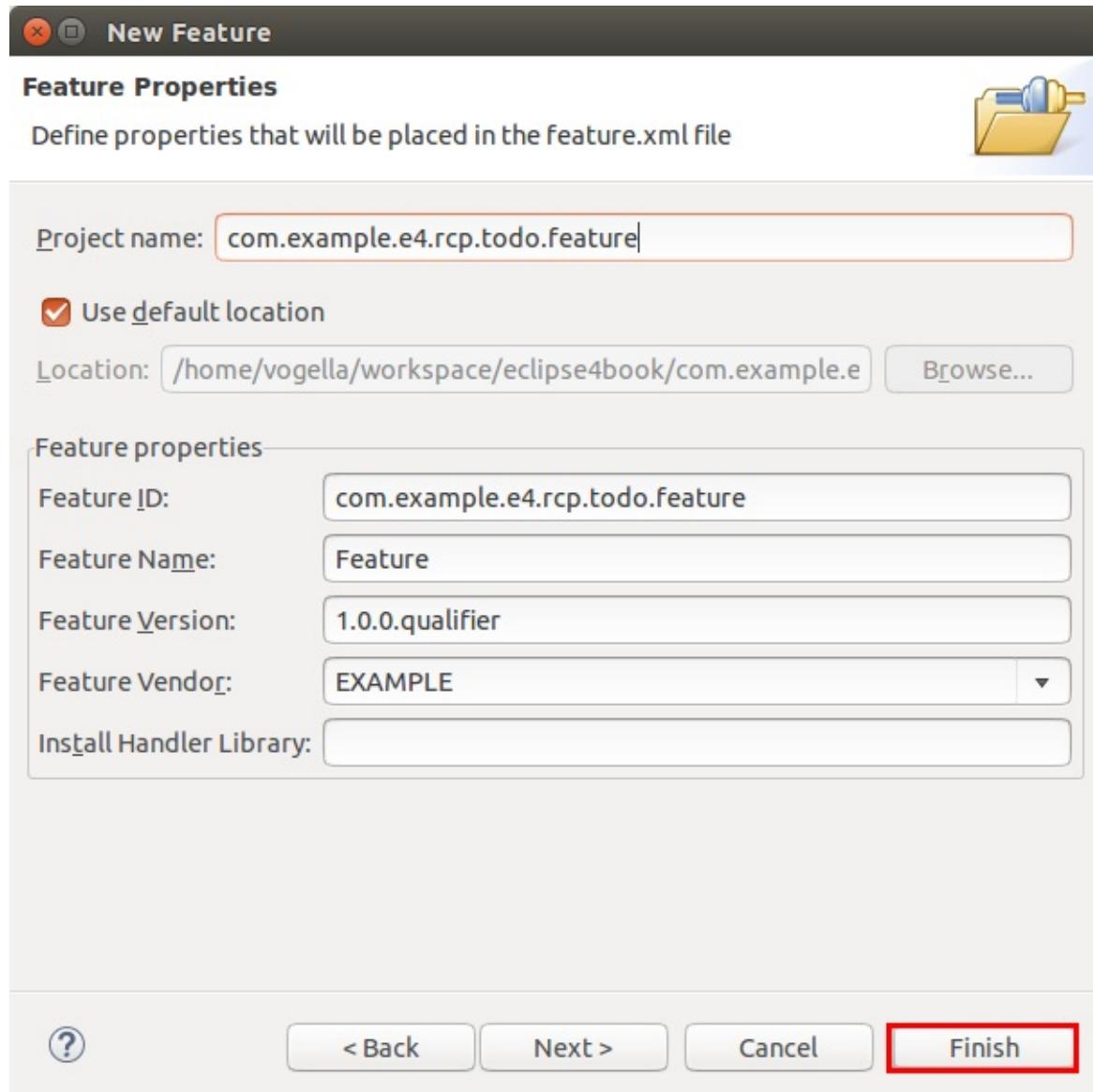


如果你是通过 Eclipse IDE 来开发、启动和输出你的产品的话，这一步是不需要的。但是对于命令行构建系统，例如 Maven/Tycho，需要你明确的设置启动级别，因此最好还是配置它们为好。

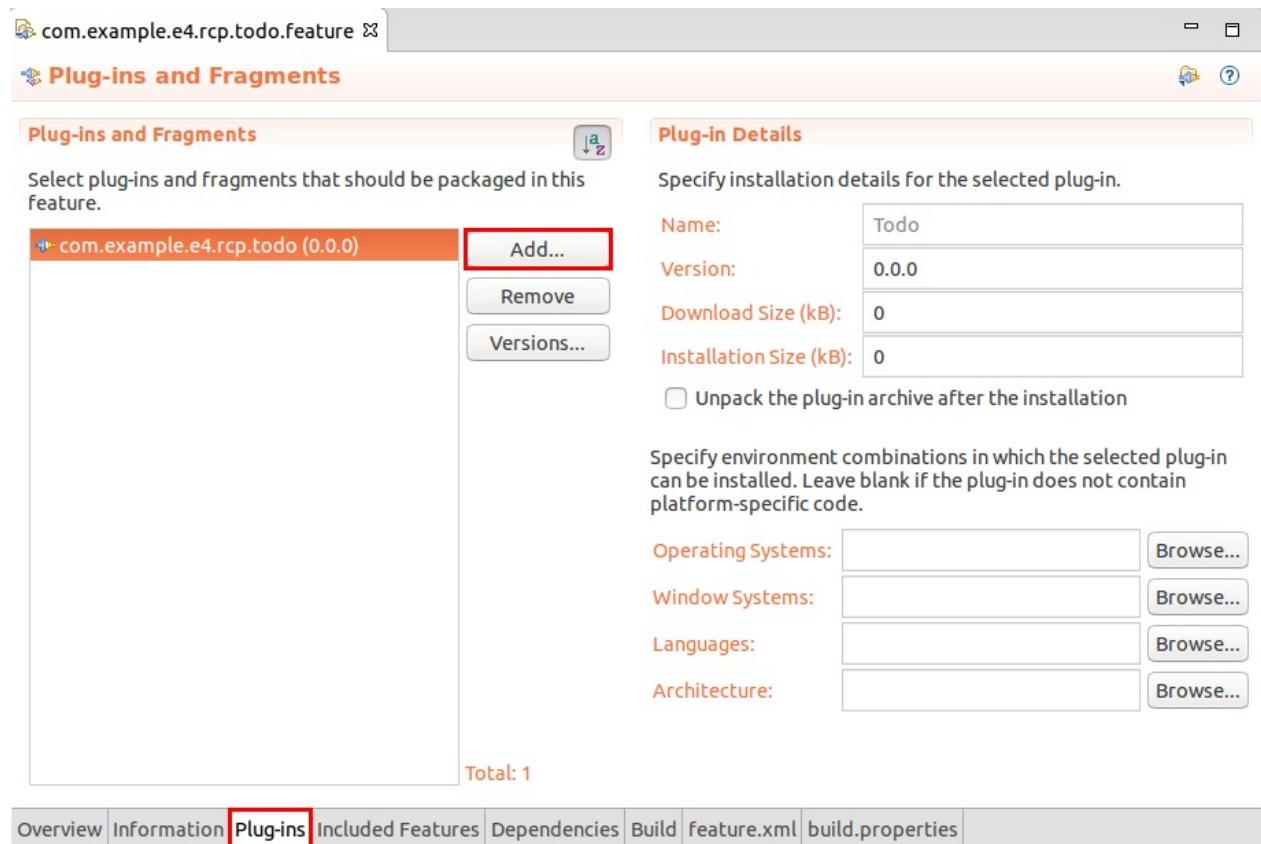
建立一个 Feature 项目

通过 文件 → 新建 → 其它... → 插件开发 → feature 项目 建立一个叫做 com.example.e4.rcp.todo.feature 的 feature 项目。

你可以在第一个向导页点击 完成 按钮。



然后在 feature.xml 文件编辑器中选择 Plug-ins 标签，点击 添加... 按钮，然后包含 com.example.e4.rcp.todo 插件到本 feature。



警告：确保你在 `Plug-ins` 标签中加入插件，在本练习题中用 `Dependencies` 标签是错误的。

在产品中输入Feature依赖

打开你的 `todo.product` 产品文件，然后改变产品配置文件来使用 features。

在产品编辑器的 `Overview` 标签中选择 `features` 选项。

The screenshot shows the Eclipse4 RCP Product Configuration interface. The title bar says "todo.product". The main area is the "Overview" tab.

General Information

This section describes general information about the product.

- ID:** [empty input field]
- Version:** [empty input field]
- Name:** to-do

The product includes native launcher artifacts

Product Definition

This section describes the launching product extension identifier and application.

- Product:** com.example.e4.rcp.todo.product
- Application:** org.eclipse.e4.ui.workbench.swt.E4Application

The [product configuration](#) is based on: plug-ins features

Testing

- [Synchronize](#) this configuration with the product's defining plug-in.
- Test the product by launching a runtime instance of it:
 - [Launch an Eclipse application](#)
 - [Launch an Eclipse application in Debug mode](#)

Exporting

Use the [Eclipse Product export wizard](#) to package and export the product defined in this configuration.

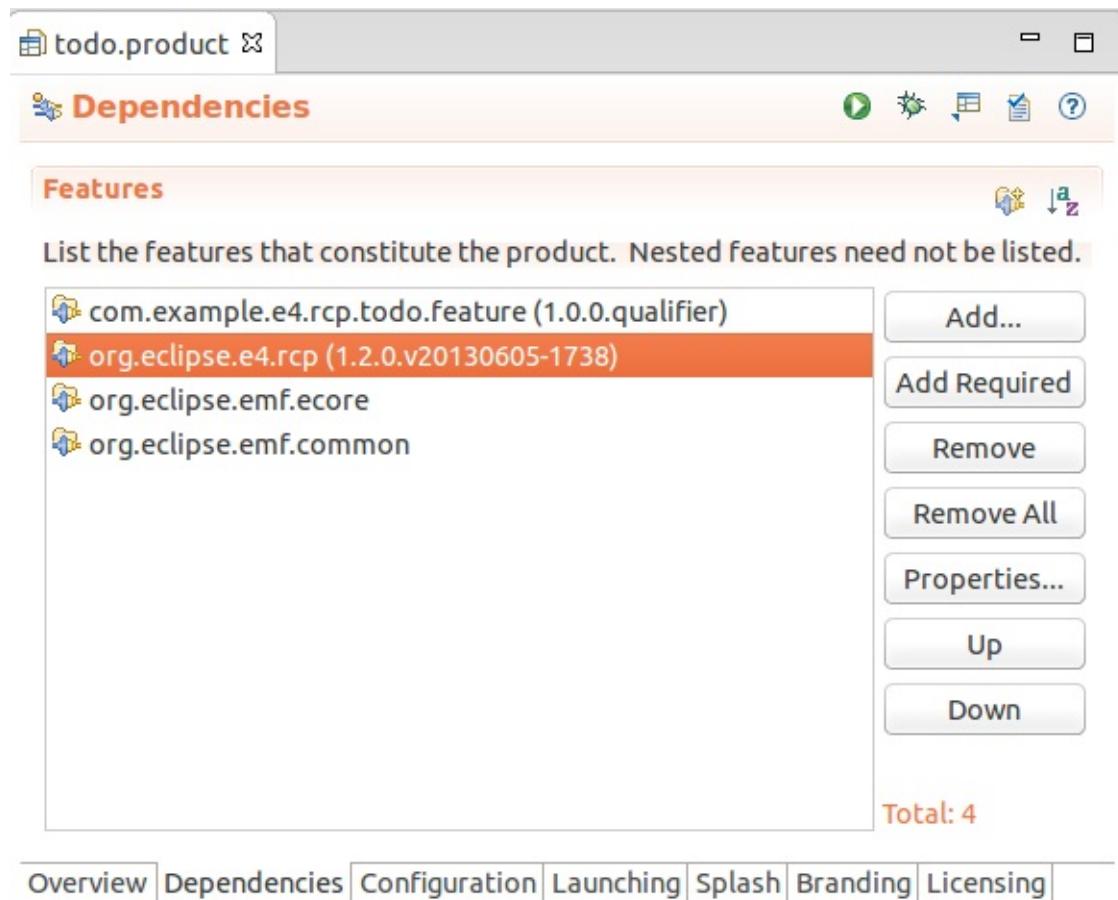
To export the product to multiple platforms:

- Install the RCP delta pack in the target platform.
- List all the required fragments on the [Dependencies](#) page.

[Overview](#) [Dependencies](#) [Configuration](#) [Launching](#) [Splash](#) [Branding](#) [Licensing](#)

选择 `Dependencies` 标签，然后通过 `添加...` 按钮添加下列的依赖。

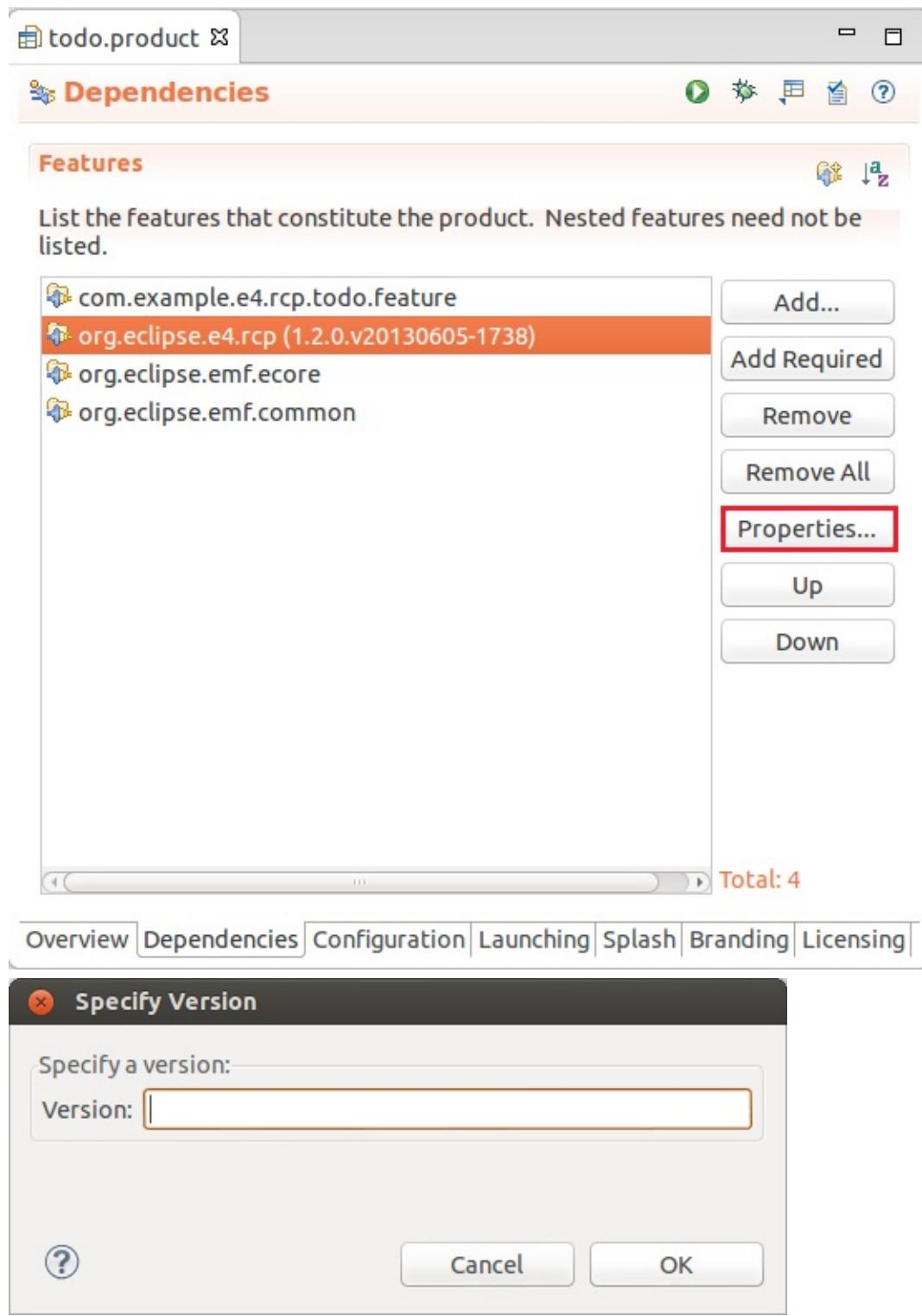
- com.example.e4.rcp.todo.feature
- org.eclipse.e4.rcp
- org.eclipse.emf.ecore
- org.eclipse.emf.common



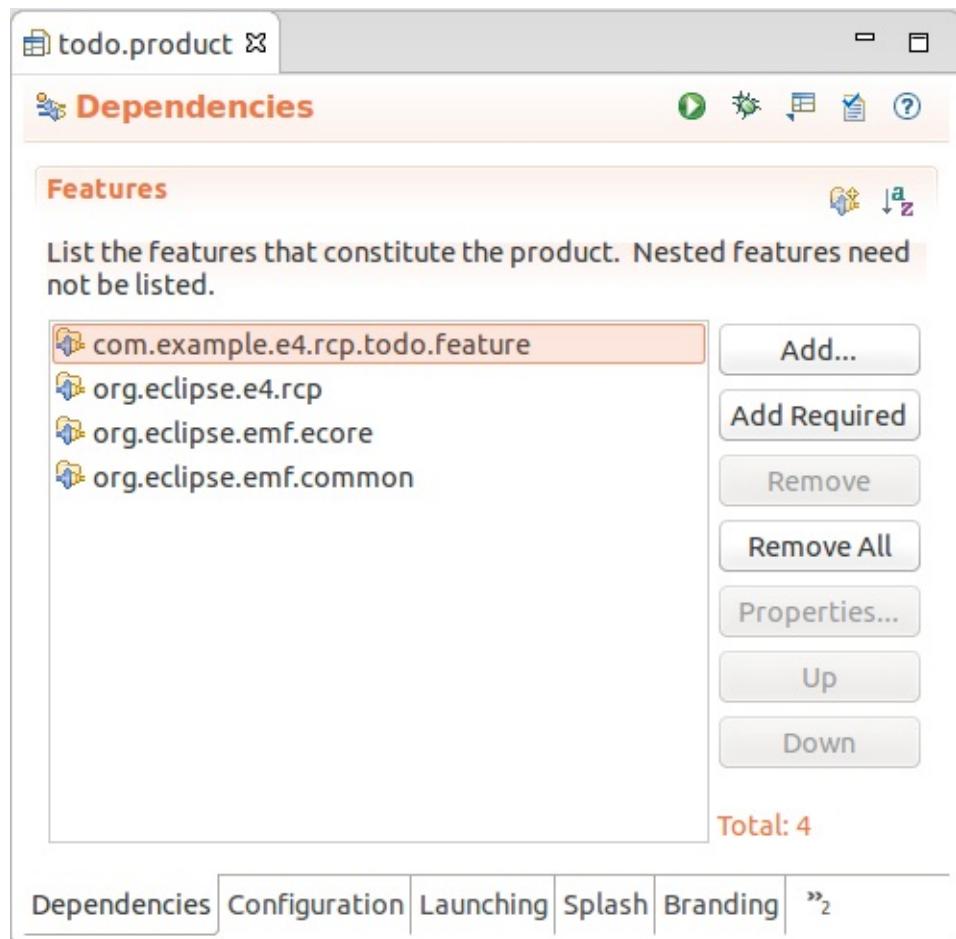
注意：如果你不能添加上列中的依赖，确保你已经修改你的产品是基于 features 的。

从产品的 features 中删除版本依赖

为了避免 `org.eclipse.e4.rcp` feature 不同版本的依赖问题，从你的产品中删除版本号。你可以通过产品配置编辑器中依赖标签中的 `Properties...` 按钮来完成。

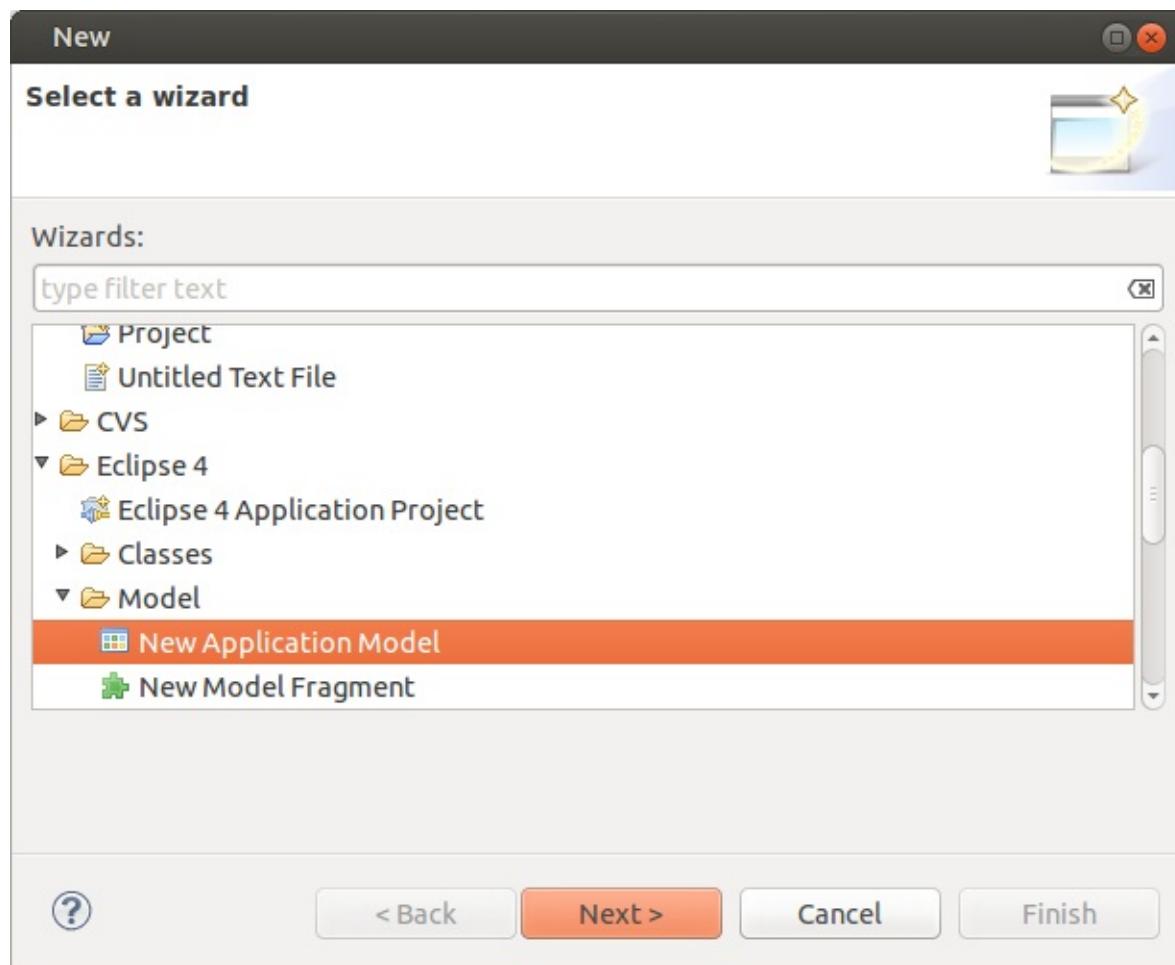


如果其他 features 有版本依赖，也删除它们。结果应该看起来类似下面的屏幕截图。

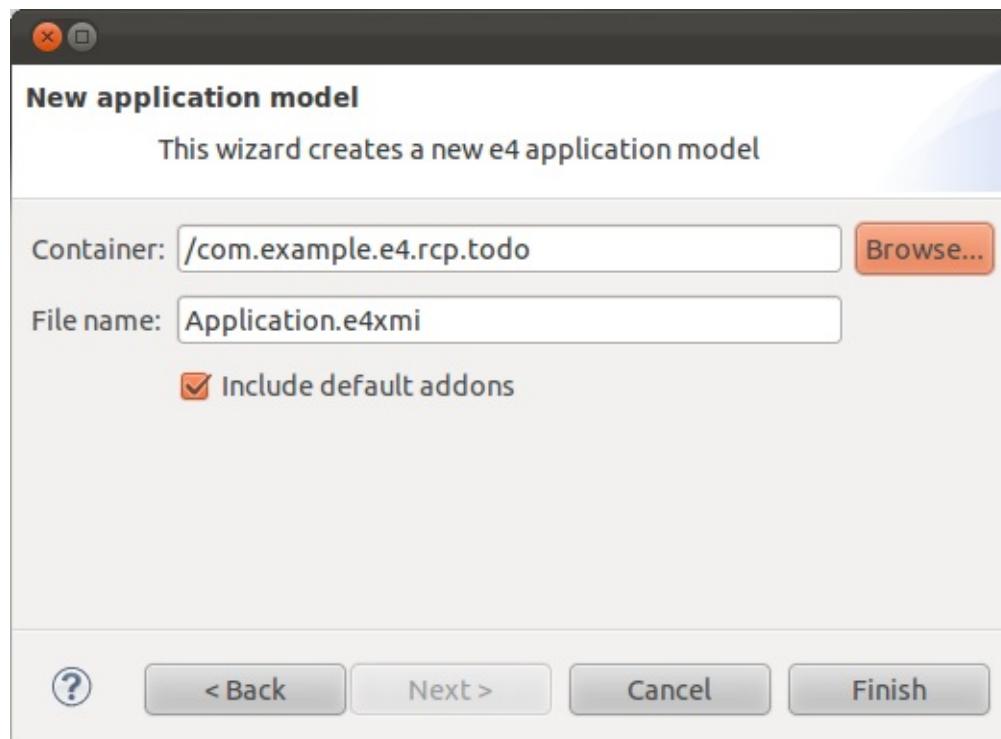


建立一个应用程序模型

选择文件 → 新建 → 其它... → Eclipse 4 → 模型 → 新建应用程序模型 来打开一个向导来建立应用程序模型文件。



输入你的 `com.example.e4.rcp.todo` 应用程序插件作为容器，然后用向导建议的文件名。

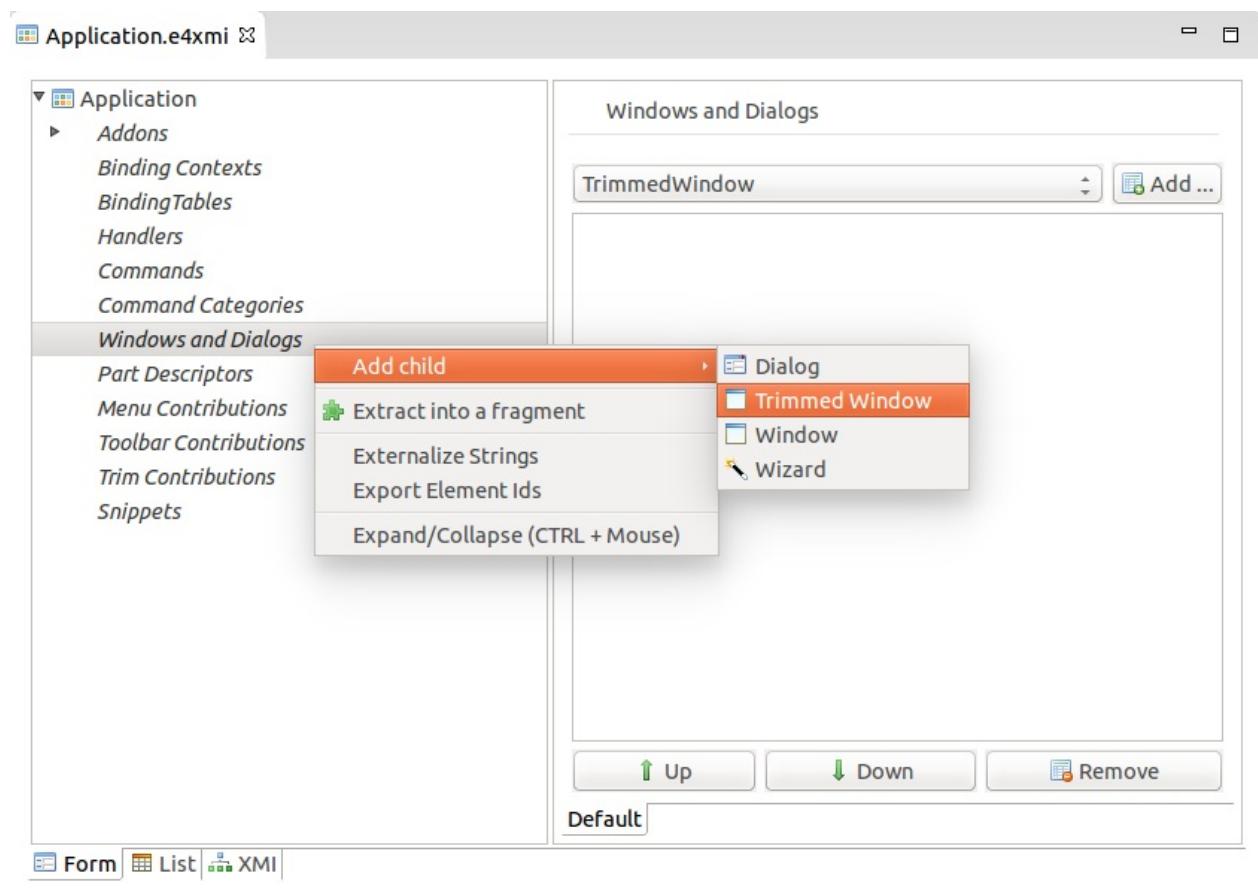


点击 完成 按钮，这将在 `com.example.e4.rcp.todo` 内建立 `Application.e4xmi` 文件，并且打开它。

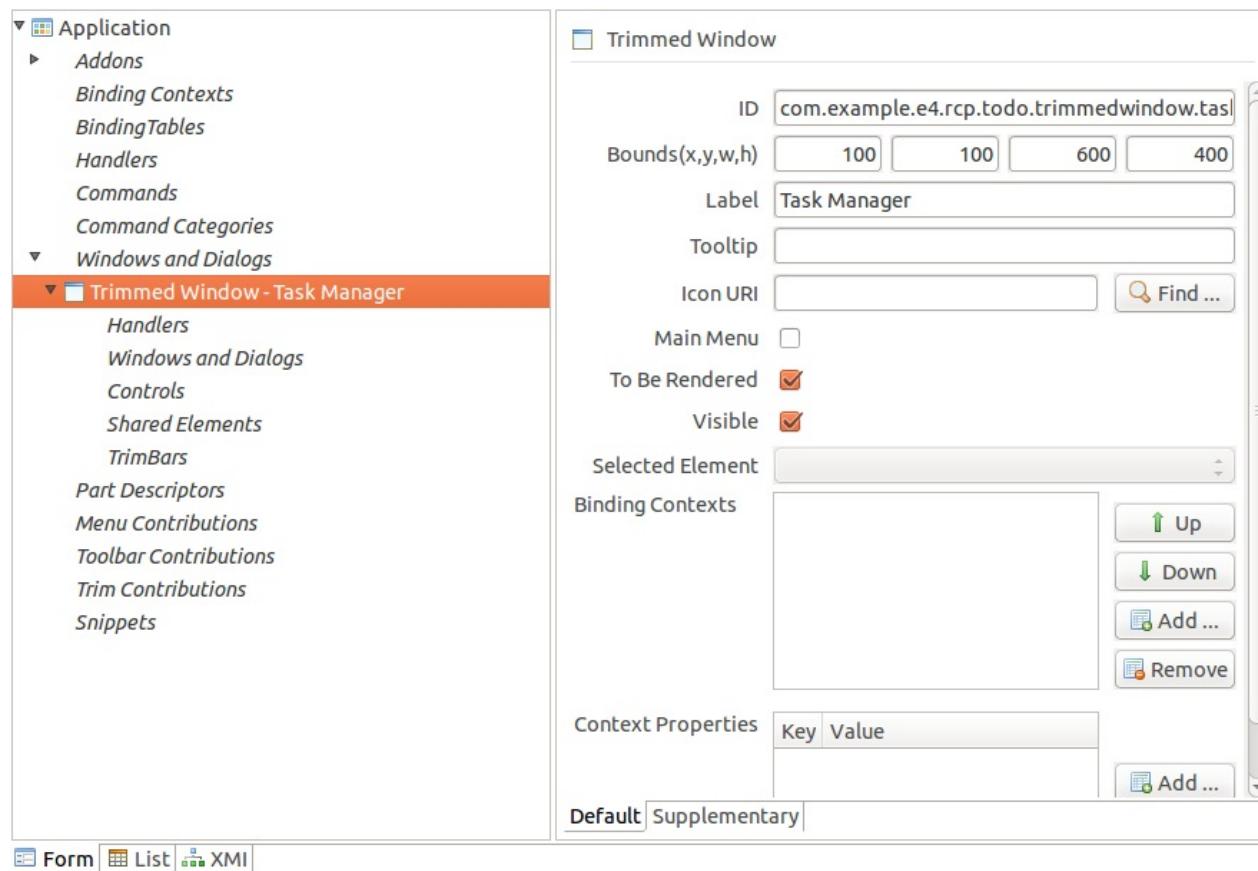
在应用程序模型中添加窗口

添加一个窗口到你的应用程序模型，这样你才有一个可视组件。

在窗口和对话框节点上右击，然后选择 Trimmed Window。



输入 ID, 位置以及窗口大小, 以及一个标签, 如下面的截图。



提示，如果你启动然后关闭应用程序，最后的应用程序状态将被框架持久化，并在下次启动的时候恢复。在开发过程中，这个是不受欢迎的特征，在第 16.1 节，[在启动时删除持久话用户改变](#)，你将修改你的产品在启动时删除所有的持久化改变。

启动应用程序

打开产品文件，然后选择 Overview 标签，在测试节点击 [启动 Eclipse 应用程序](#)。

Testing

1. [Synchronize this configuration with the product's defining plug-in.](#)
2. Test the product by launching a runtime instance of it:
 - [Launch an Eclipse application](#)
 - [Launch an Eclipse application in Debug mode](#)

[Overview](#) [Dependencies](#) [Configuration](#) [Launching](#) [Splash](#)

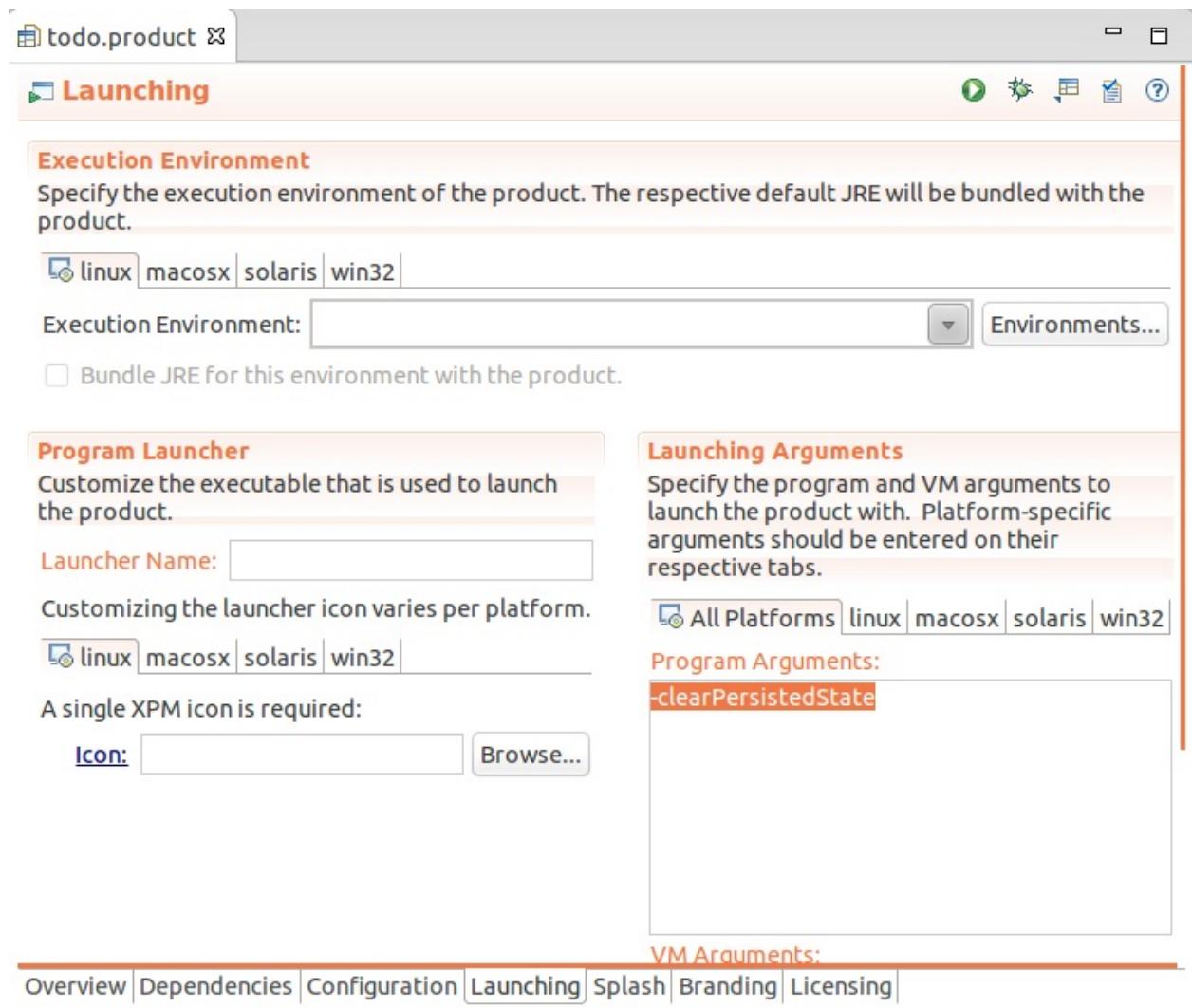
验证你的应用程序启动了，你应该看到一个空窗口，可以被移动、改变大小、最小化、最大化，以及关闭。

练习：删除持久化模型数据

在启动时删除持久化模型数据

为了确保总是使用你的应用程序模型的最后版本，在你的产品配置文件中加入 `-clearPersistedState` 参数。

下图显示在产品配置文件中的配置。



为何需要这个设置

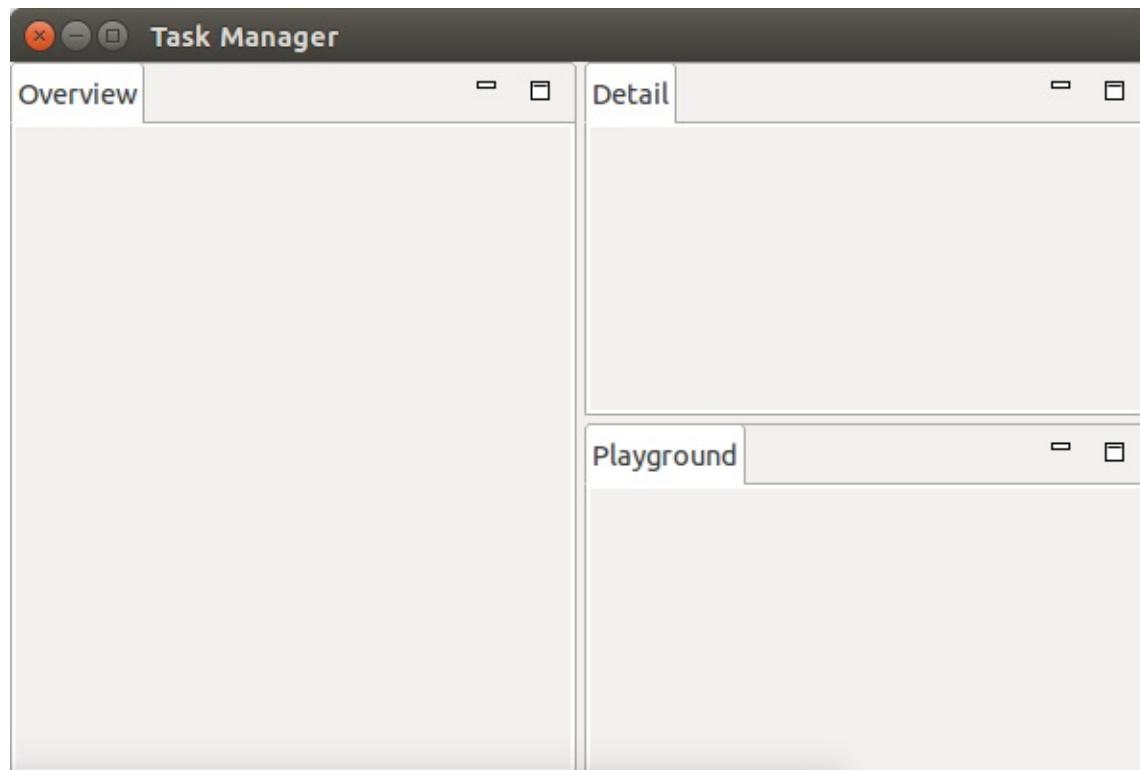
在Eclipse应用程序启动时，Eclipse平台恢复应用程序最后退出时的状态，在开发期间这会导致改变没有正确的应用和显示，例如，你定义了一个新的菜单项，但是这个菜单项没有在你的应用程序中显示。

作为替代方案，你也可以在你的 Run 配置的 Main 标签设置 clear 标志。这也会删除其它持久化的数据，例如首选项值。

练习：用户接口建模

期望的用户接口

在下面的练习中，你将建立你的应用程序的基本用户接口，在本练习的最后，你的用户接口将看起来如下：



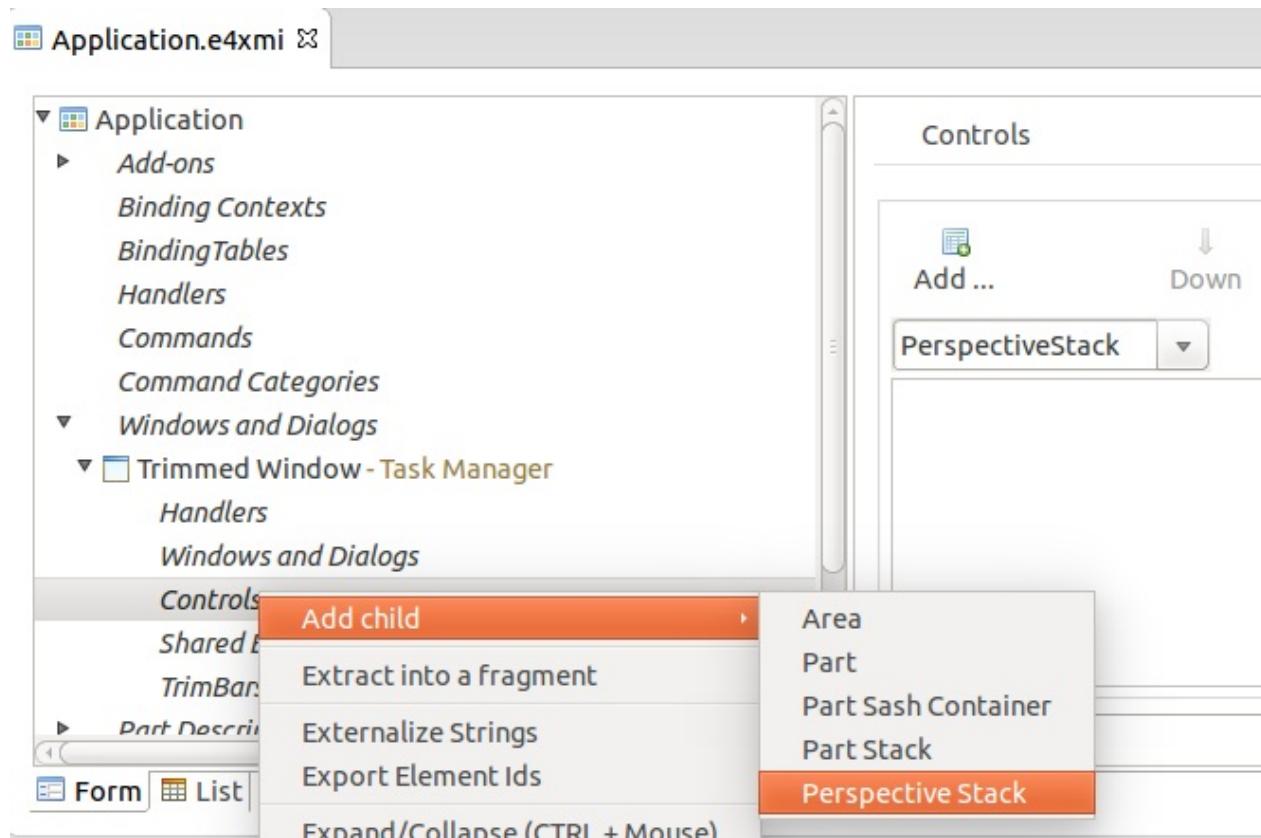
打开 Application.e4xmi 文件

在 Application.e4xmi 文件上双击或右击，然后选择 Open With → Eclipse 4 模型编辑器 来打开 Application.e4xmi 文件。

添加一个透视图

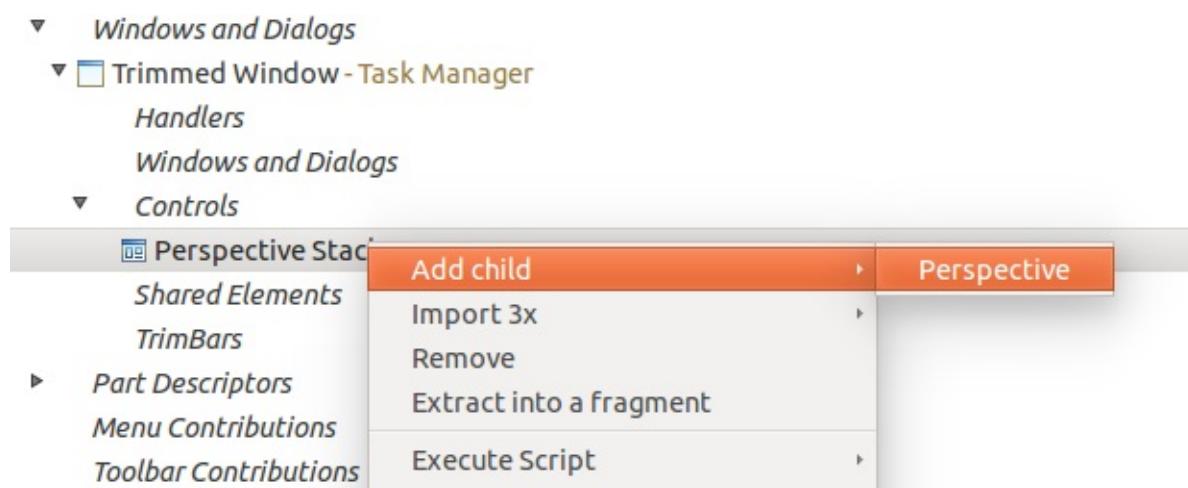
添加一个带有一个透视图的透视图找到你的应用程序，后面你可以轻易的添加更多。

在 Application.e4xmi 文件中导航到你的窗口，选择 Controls 节点，通过 Controls 项的上下文菜单添加透视图栈，如下图：

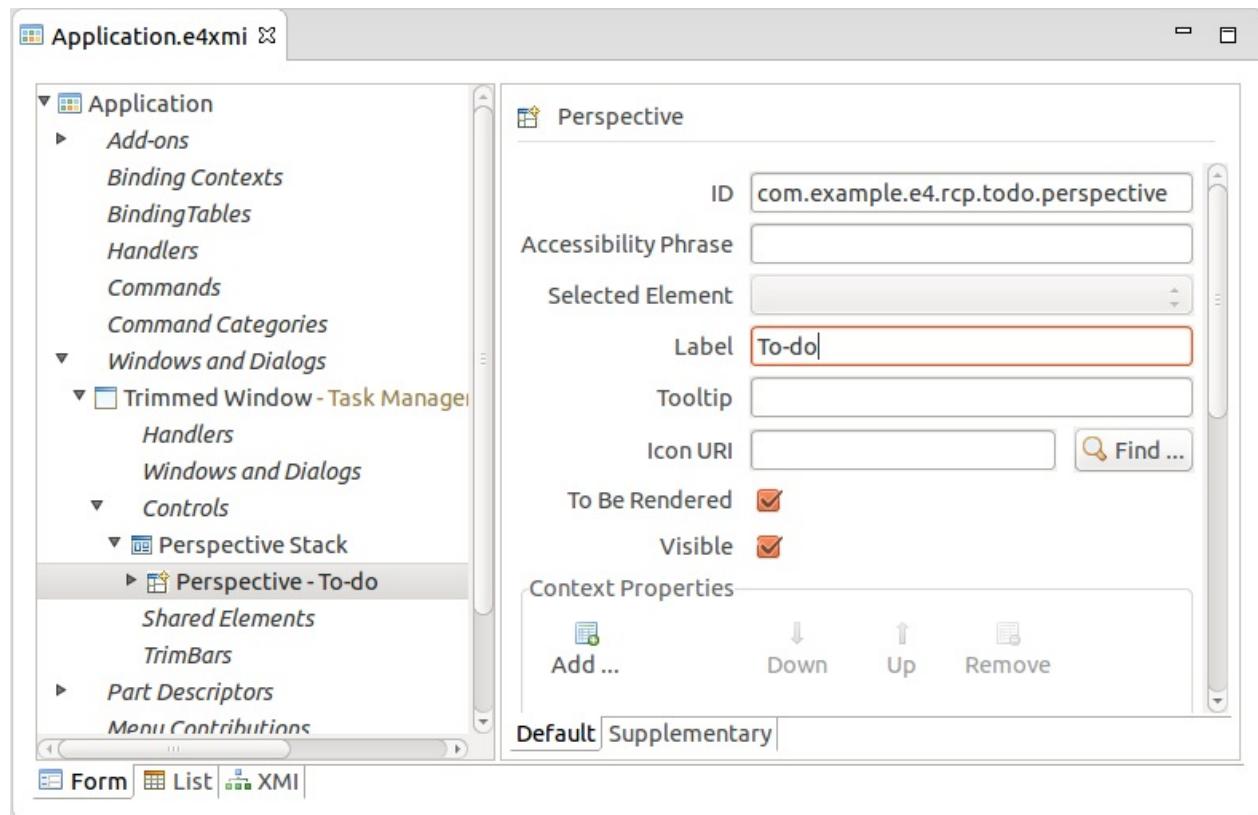


提示：作为上下文菜单的替代方法是用 detail 页的 **添加...** 按钮来为选择的元素添加子元素。

在建立透视图栈之后，添加一个透视图到里面，要么通过上下文菜单，要么通过 **添加...** 按钮。



在 Label 字段输入 `To-do`，在 ID 字段输入 `com.example.e4.rcp.todo.perspective`。

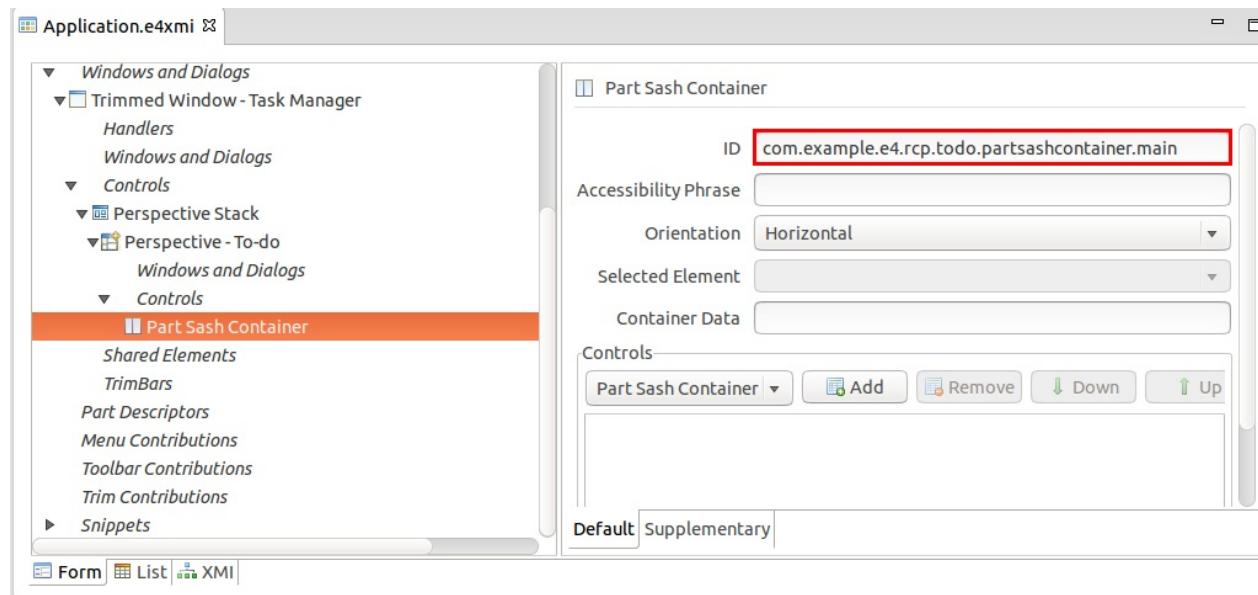


添加 Part Sash 以及 Part Stack 容器

选择在新建的透视图下面的 Controls，然后添加一个 Part Sash 容器元素。

Controls	Add child	Part Sash Container
Shared Elements	Extract into a fragment	Part Stack
TrimBars	Externalize Strings	Part
Part Descriptors		Area

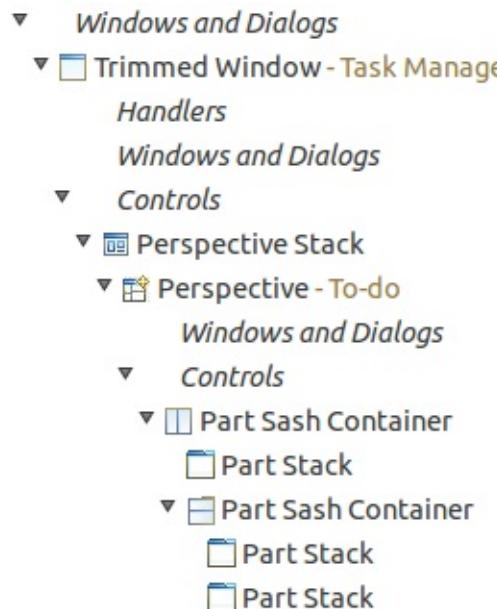
修改它的方向属性为水平，然后在 ID 字段输入 com.example.e4.rcp.todo.partsashcontainer.main。



添加一个 Part Stack 作为 Part Sash 的第一个子元素。

重新选择父 Part Sash 容器，然后添加另一个 Part Sash 容器元素，现在添加了 2 个 Part Stack 到二级 Part Sash 容器。

经过这些修改之后，你的应用程序模型看起来应该类似下面的截图：

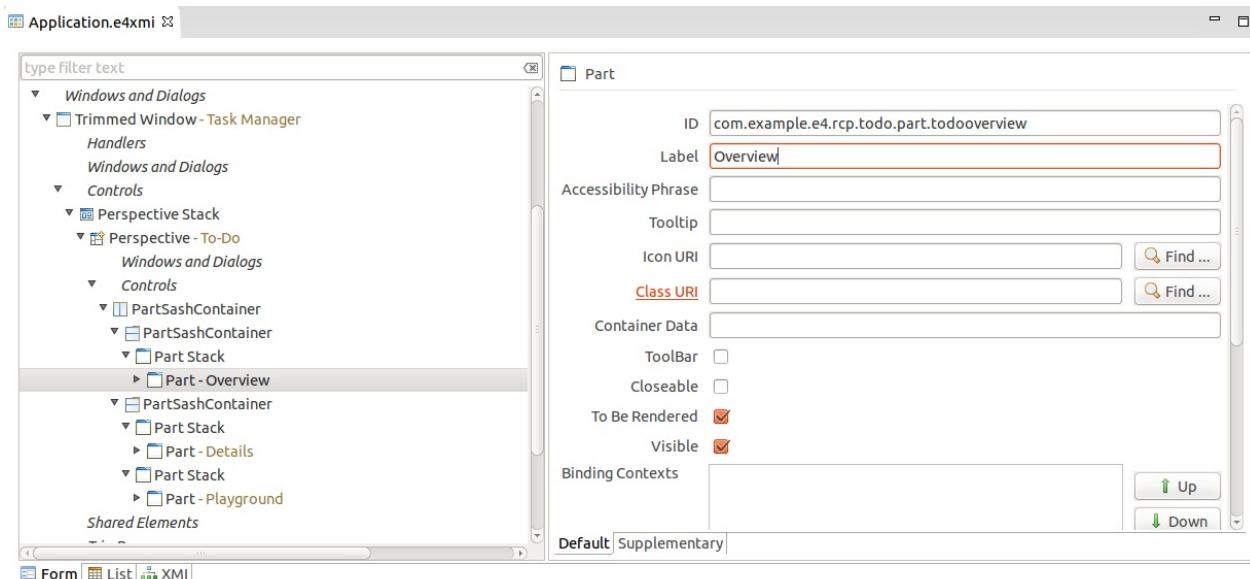


建立 Part

为每个 Part Stack 添加一个 Part 模型元素，Part 的 ID 前缀使用 com.example.e4.rcp.todo.part，后缀使用下表中的 Label 填充到 Part 编辑器中的相应字段。

ID 后缀	Label
.todooverview	Overview
.tododetails	Details
.playground	Playground

最后你的应用程序模型的结构应该类似下面的截图，这个截图也显示了 Overview Part 的详细数据。



验证用户接口

启动你的产品，然后验证用户接口是不是预期效果，如果需要的话重新排列模型元素。模型编辑器支持拖放来重新排列。

也要注意你已经看到了结构，但到目前为止还没有建立任何的 Java 类。

练习：连接 Part 和 Java 类

建立一个新包以及一些 Java 类

在应用程序插件中建立 `com.example.e4.rcp.todo.parts` 包。

在这个包中建立 3 个 Java 类，分别叫做 `TodoOverviewPart`，`TodoDetailsPart`，`PlaygroundPart`。

提示：你可以通过模型编辑器中细节面板中点击 Class URI 超链接来建立类，这会连接建立的类到模型对象。如果你这样做，你可以跳过本练习中的小节 [连接 Parts 与 Java 类](#)。

下面的代码显示了 `TodoDetailsPart` 类，所有类不应该从其他类派生，也不要实现任何接口。

```
package com.example.e4.rcp.todo.parts;

public class TodoDetailsPart {
```

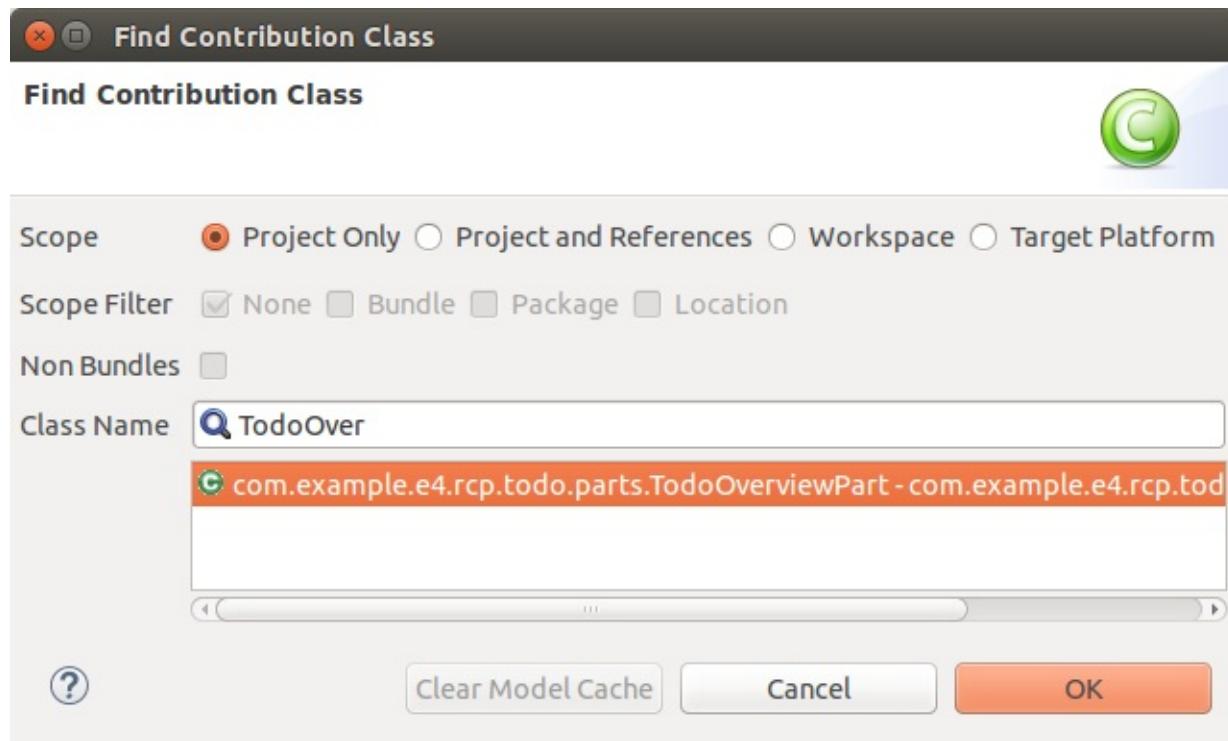
连接 Parts 与 Java 类

打开 `Application.e4xmi` 文件，然后连接类与对应的 Part 模型元素。你可以通过 Part 模型元素的 Class URI 属性来完成。

下表给出了哪个元素需要连接的总览。

Class	Part ID 后缀
<code>TodoOverviewPart</code>	<code>*.todooverview</code>
<code>TodoDetailsPart</code>	<code>*.tododetails</code>
<code>PlaygroundPart</code>	<code>*.playground</code>

Eclipse 4 模型编辑器允许你通过 `Find...` 按钮查询存在的类，最初的 Contribution Classes 列表是空的，在 Class Name 字段中输入来看看结果。



下面的截图显示了 Overview Part 的结果。

Part

ID	com.example.e4.rcp.todo.part.todooverview
Label	Overview
Accessibility Phrase	
Tooltip	
Icon URI	
<u>Class URI</u>	bundleclass://com.example.e4.rcp.todo/com.example.e4.rcp.todo.parts.TodoOverviewPart
Container Data	
ToolBar	<input checked="" type="checkbox"/>
Closeable	<input checked="" type="checkbox"/>
To Be Rendered	<input checked="" type="checkbox"/>
Visible	<input checked="" type="checkbox"/>

验证

运行你的应用程序，它应该可以启动，但是用户接口看起来应该和之前没有不同。

为了验证模型对象被 Eclipse 运行时建立了，为某个类添加一个无参数的构造器，然后加入一行 `System.out.println()` 语句，验证启动应用程序时，构造器被调用了。

输入依赖

添加插件依赖

在接下来的练习中，你将用到其他 Eclipse 插件的功能，这需要你在应用程序中定义这些插件的依赖关系。

记住应用程序插件代表 `com.example.e4.rcp.todo` 插件。

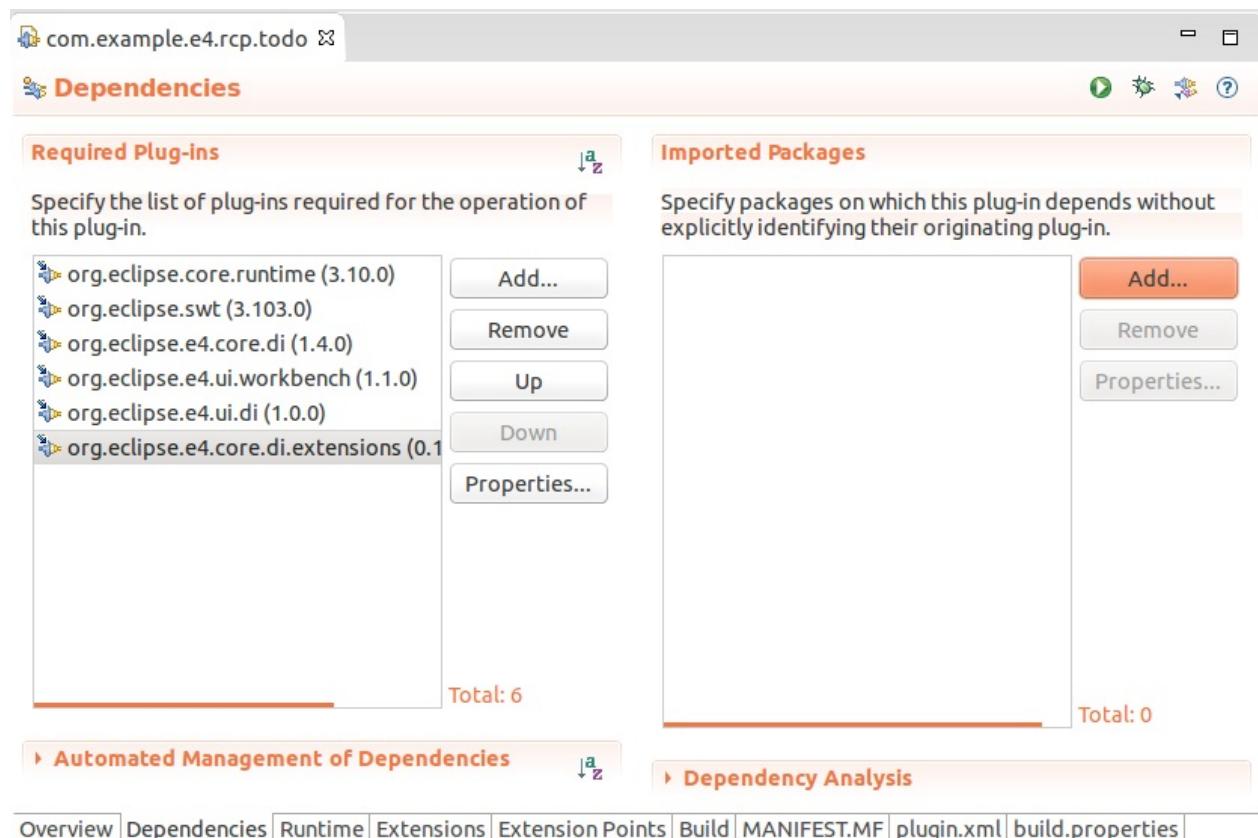
打开 `META-INF/MANIFEST.MF` 文件，然后选择 `Dependencies` 标签，在 `需求的插件` 中用 `添加...` 按钮来添加下列的插件作为依赖。

- `org.eclipse.core.runtime`
- `org.eclipse.swt`
- `org.eclipse.e4.core.di`
- `org.eclipse.e4.ui.workbench`
- `org.eclipse.e4.ui.di`
- `org.eclipse.e4.core.di.extensions`

注意在 Eclipse 4.4 之前，你必须定义 `javax.annotation` 作为包依赖。Eclipse 4.4 中，`javax.annotation` 包通过 `org.eclipse.core.runtime` 插件重新输出了。还有 `javax.inject` 插件也通过 `org.eclipse.core.runtime` 重新输出了，所以不需要定义对它的依赖。

验证

结果应该类似于下列的截图：



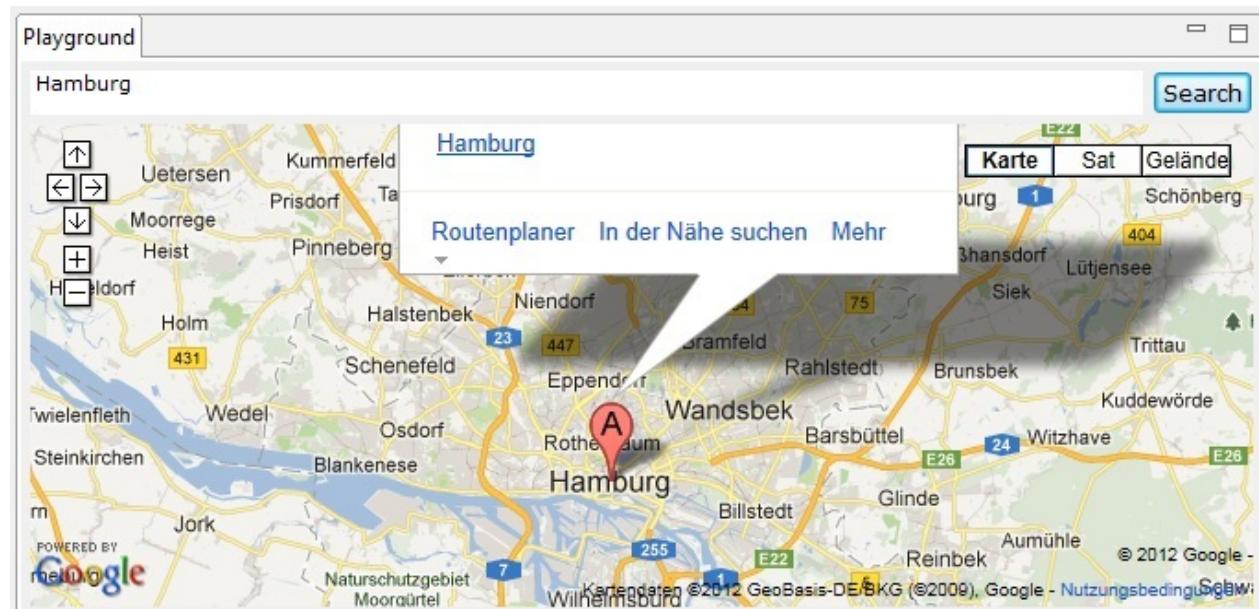
练习：用 Swt 浏览器插件

实现

本练习将在 SWT 浏览器控件中显示 Google 地图。

注意：本练习在 Linux 系统上可能工作不正常，因为浏览器控件的某些 Linux 版本不能正常工作，参考 Eclipse SWT FAQ How do I use the WebKit renderer on Linux-GTK 的答案。

修改 PlaygroundPart 类，让 Part 看起来如下：



注意：如果 Google 修改了它的 API 的话，本例子也可能不能工作。

如果你在 text 字段输入文字，然后点击按钮，地图应该基于 text 字段中的输入居中，输入应该为城市。

方案

PlaygroundPart 类的代码看起来如下：

```
package com.example.e4.rcp.todo.parts;

import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;

import javax.annotation.PostConstruct;

import org.eclipse.e4.ui.di.Focus;
import org.eclipse.swt.SWT;
import org.eclipse.swt.browser.Browser;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;

public class PlaygroundPart {
    private Text text;
    private Browser browser;
```

```

@PostConstruct
public void createControls(Composite parent) {
    parent.setLayout(new GridLayout(2, false));

    text = new Text(parent, SWT.BORDER);
    text.setMessage("Enter City");
    text.setLayoutData(new GridData(SWT.FILL, SWT.CENTER, true, false, 1, 1));

    Button button = new Button(parent, SWT.PUSH);
    button.setText("Search");
    button.addSelectionListener(new SelectionAdapter() {
        @Override
        public void widgetSelected(SelectionEvent e) {
            String city = text.getText();
            if (city.isEmpty()) {
                return;
            }
            try {
                // not supported at the moment by Google
                // browser.setUrl("http://maps.google.com/maps?q="
                // + URLEncoder.encode(city, "UTF-8")
                // + "&output=embed");
                browser.setUrl("https://www.google.com/maps/place/"
                    + URLEncoder.encode(city, "UTF-8")
                    + "/&output=embed");

            } catch (UnsupportedEncodingException e1) {
                e1.printStackTrace();
            }
        }
    });
}

browser = new Browser(parent, SWT.NONE);
browser.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true, 2, 1));
}

@Focus
public void onFocus() {
    text.setFocus();
}
}

```

在文本框中收入 new york 试试。

依赖注入介绍

See [Dependency injection in Java](#) for an introduction into the concept of dependency injection,

依赖注入与注解

在 Eclipse 中定义类依赖

Eclipse 编程模型支持符合 Java 规范请求 330(JSR330)的构造器、方法以及字段注入。Eclipse 还定义了附加的 annotations 用作依赖注入，大部分重要的 annotations 在下一节描述，其它更特殊的 annotations 在相应的章节描述。

Eclipse 依赖框架确保 key 以及注入对象的类型是正确的，例如，如果你希望对 xyz key 使用 Todo 类型，就像下面的字段声明那样，框架将只有在找到一个可指派的类型时才会注入对象。

```
@Inject @Named("xyz") Todo todo;
```

在 Eclipse 中 annotations 定义类依赖

下表给出了基于 JSR330 以及 Eclipse 独有的依赖注入相关的 annotations。

Annotation	描述
@javax.inject.Inject	在 JSR330 中定义，可以添加到字段、构造器、方法上。Eclipse 框架尝试注入相应的对象到字段或实例的参数。
@javax.inject.Named	在 JSR330 中定义，为应该被注入的对象定义 key。通常，全资格类名用作 key。默认值的几个 keys 是在 IServiceContants 接口中定义为常数。
@Optional	Eclipse 特殊的 annotation，标示一个注入值是可选的。如果对于给定的 key 没有有效的对象，框架不会抛出一个异常。 具体的行为依赖 @Optional 放在什么地方。下面的描述基于 key，如果 key 不能被解决，下面情况将发生： 1. 对于参数，将注入一个 null 值； 2. 对于方法，方法调用被跳过； 3. 对于字段，值不会注入； 注意 null 是一个上下文可接受的值，并且 key 从上下文删除时不同。例如，如果下面是调用了 context.set(SOMEKEY, null)，所有监听 SOMEKEY 的将被注入 null。
@GroupUpdates	Eclipse 特殊的 annotation，指示为这个 @Inject 更新应该批处理。如果你在 Eclipse 上下文中改变了这样的对象，更新是通过 IEclipseContext 对象的 processWaiting() 方法来触发的。这个 annotation 的主要用途是为平台提供性能优化，应该很少在 RCP 应用程序中使用。

注意 Eclipse 平台支持其他特殊用途的 annotations，例如，为接受事件（通过事件服务器发送）或在 preferences 上工作。对于所有定义在 Eclipse 平台的标准 annotations 的摘要参考 ???。

Eclipse 在哪个对象上执行依赖注入？

Eclipse 运行时通过应用程序模型引用的 Java 类来创建对象。在这个实例化过程中，Eclipse 运行时扫描类定义中的 annotations，基于这个 annotations，Eclipse 框架执行注入。

Eclipse 不会在你的代码中通过 new 操作符创建的对象上自动执行依赖注入。

基于 key/value 改变的动态依赖注入

Eclipse 框架追踪哪个对象依赖到哪个 key 和值，如果 key 指向的值改变了，Eclipse 框架重新注入新的值，这意味着应用程序可以不用安装（以及删除）监听器。

例如，你可以通过 @Inject 定义你希望得到当前选择的对象，如果选择改变了，Eclipse 框架将注入新的值。

重新注入只会在标注了 @Inject 的方法和字段上工作，它不会在标记了 @PostConstruct 的构造器和方法的参数上工作，因

为这些方法只会执行一次，这将在后面的"Eclipse 上下文生命周期"中解释。

注意这不是说 Eclipse 跟踪 key 指向的值的字段，例如，如果 mykey1 指向一个 Todo 对象，然后 mykey1 指向一个新对象，这将触发重新注入到所有相关类依赖的对象，但是如果字段内部的字段改变了，它不会触发重新注入。

Eclipse 上下文

什么是 Eclipse 上下文？

在 Eclipse 应用程序启动的时候，Eclipse 运行时基于 `IEclipseContext` 接口创建一个对象，这个对象叫做 Eclipse 上下文。

上下文类似于 Map 数据结构，对象可以与特定的 key 关联，key 是字符串，在几个情况下使用全资格类名作为 key。值（key指向的）可以被注入到其他对象。

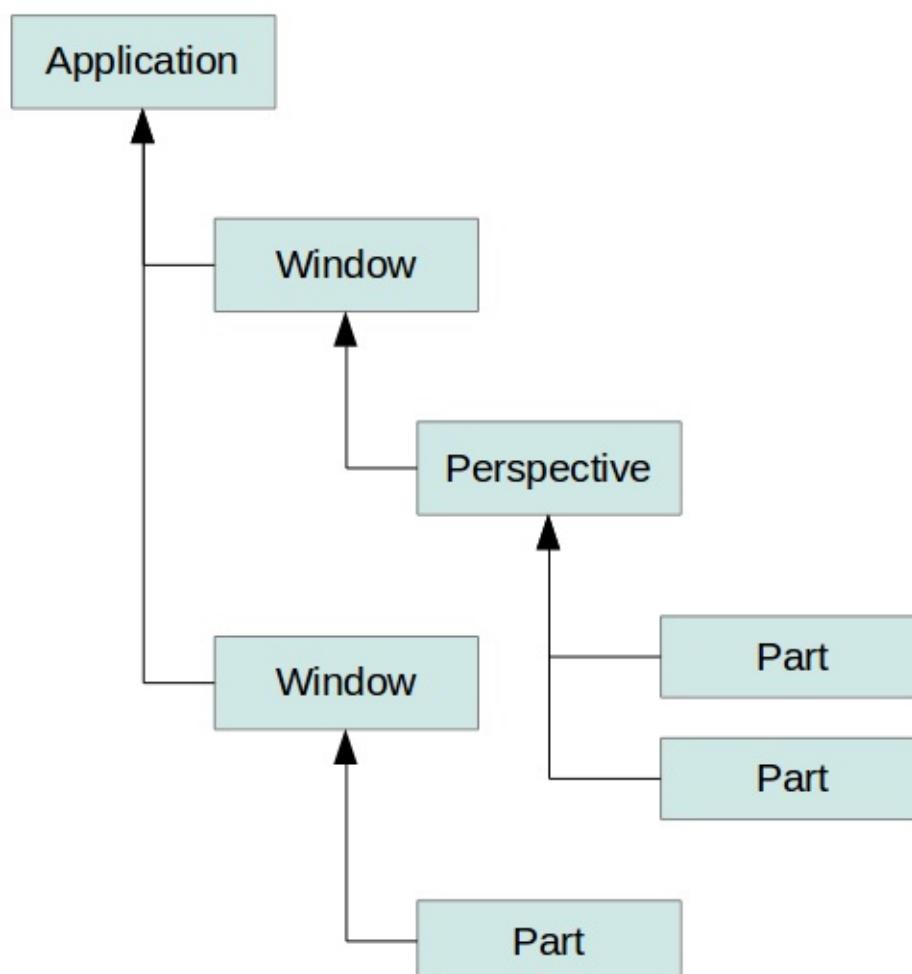
Eclipse 上下文中的关系定义

就像前面描述的，Eclipse 上下文类似 Map 数据结构，但是和 Map 不同，Eclipse 上下文是层次结构，并且也可以为请求的 key 动态计算值。

对于特定的模型对象（参考“哪个模型对象有一个本地的上下文？”），一个本地上下文被创建，这个上下文关联一个应用程序模型对象。

不同的上下文对象是通过一个基于你的应用程序模型的层次树结构来连接，这个层次中的最高层是应用程序上下文。

一个例子上下文层次可以用下图来描绘。



对象可以放在不同的层中，这允许同样的 key 指向层次中不同的对象。

例如，一个 Part 可以通过一个类似于下面的字段声明表达一个依赖到它的 Composite 对象：`@Inject Composite parent;`，因 Eclipse 上下文

为 Part 有不同的本地上下文，它们可以得到不同 Composite 对象。

哪个模型对象有一个本地上下文？

当前下列的模型元素实现 MContext 接口，因此有它们自己的上下文：

- MApplication
- MWindow
- MPerspective
- MPart
- MPopupMenu

Eclipse 上下文的生命周期？

Eclipse 在启动时根据应用程序模型建立上下文层次，通常，它将预定义 keys 的特定对象放到上下文中，例如，services 来控制 Eclipse 框架的功能。

模型对象以及基于 class URI 属性创建的对象是通过 Eclipse 平台创建。对于每个有定制上下文的模型元素，Eclipse 框架决定哪个对象应该是模型对象有效的本地上下文，如果需要的话，它也创建模型元素类 URI 属性引用的 Java 对象，这是，例如，如果 Part 对用户可见的情况。

Note The renderer framework is responsible for creating the local context of the UI related model elements. This framework allows you to define classes which are responsible for setting up the UI implementation of the model objects. A class responsible for a model element is called the renderer for this model element. For example, the ContributedPartRenderer class is the default renderer for part model objects. This renderer creates a Composite for every part and puts this Composite into the local context of the part.

在初始化创建 Eclipse 上下文层次之后，框架或应用程序代码可以改变上下文中存储的 key-value 对。这种情况下，对象是通过相关的 Eclipse 功能（例如通过 Eclipse 依赖注入框架）创建的会更新为新值。

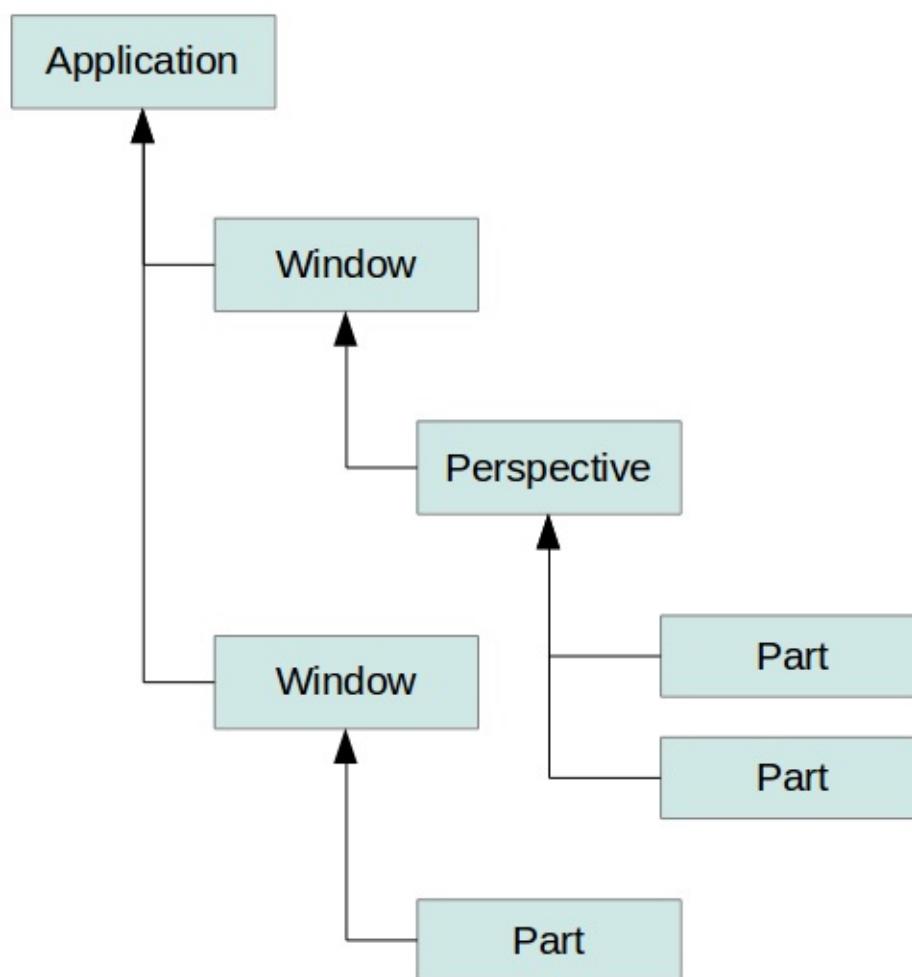
上下文中的对象在内存中保存（暂时性的），也就是说，当应用程序停止时，上下文就销毁了。

对依赖注入有效的对象

依赖注入如何选择对象？

在前面的“在 Eclipse 中定义类依赖”中，通过 Eclipse 创建的一个对象可以用 annotations 来描述他的类依赖。

在为一个 Eclipse 创建的对象依赖注入时，Eclipse 框架基于指定的 key 来查询适当的对象，查询从与应用程序模型对象关联的本地上下文开始，如果这个 key 无效，Eclipse 继续查询父上下文，这个过程一直继续，直到主上下文。



后面的章节，你讲学到 Eclipse 上下文不是注入的对象的唯一可能来源。后面的其他例子涵盖 OSGi 服务，Preferences，事件以及定制对象。

如何访问模型对象？

对于应用程序模型中引用的类，Eclipse 框架在需要的时候建立相应的对象，这样的对象可以通过依赖注入访问它对应的模型对象。

例如，在 Part 的实现中，你可以通过：`@Inject MPart part;` 访问 Part 的模型信息。

Eclipse 上下文中默认的项

Eclipse 框架中上下文中建立几个对象，它们是：

- 模型对象 - 包含应用程序模型的数据;
- services - 通过 Eclipse 平台或通过 OSGi 服务注册定义的软件组件;
- 几个其他明确添加到上下文中的对象;

上下文可以通过应用程序代码以及框架进行修改，Eclipse 框架自动的跟踪它创建的对象的依赖，它可以更新它们，参考“基于 key/value 修改动态依赖注入”。

访问活动 Part 或 Shell 的限定词

Eclipse 平台将当前选择的 Part 和活动的 shell 放在应用程序对象的 IEclipseContext 中，keys 是在 IServiceConstants 结构中定义。

例如，下面的方法将允许你在其他 Part 中跟踪当前活动的 Part。

```
// tracks the active part
@Inject
@Optional
public void receiveActivePart(@Named(IServiceConstants.ACTIVE_PART) MPart activePart) {
    if (activePart != null) {
        System.out.println("Active part changed "
            + activePart.getLabel());
    }
}
```

要跟踪活动的 shell，用 IServiceConstants.ACTIVE_SHELL key.

```
// tracks the active shell
@Inject
@Optional
public void receiveActiveShell(@Named(IServiceConstants.ACTIVE_SHELL) Shell shell) {
    if (shell != null) {
        System.out.println("Active shell (Window) changed");
    }
}
```

注意 Eclipse 用 handlers 来定义可以通过菜单或工具条触发的 actions。对于一个 handler 实现类，不需要用这些限定词，因为 handler 在应用程序活动的上下文中执行。

用 @Active 跟踪一个子上下文

@Active annotation 允许你跟踪子上下文中的值，Eclipse 框架跟踪 IEclipseContext 层次中当前活动的分支，例如，如果用户选择了一个 Part，那么在 IEclipseContext 层次中，从根到 Part 的 IEclipseContext 路径是当前活动分支。

用 @Active annotation，你可以跟踪当前活动分支中的子元素的值，当活动分支改变了，那么 key 引用的值被重新注入到使用 @Active annotation 的对象。

下面的片段演示这个 annotation 的用法：

```
public class MyOwnClass {
    @Inject
    void setChildValue(@Optional @Named("key_of_child_value") @Active String value) {
        this.childValue = value;
    }
}
```

注意，@Active annotation 当前在 Eclipse 框架本身没有用到，本书的作者也没有找到一个合适的使用情况。

用 Annotation 定义行为

API 定义

如果你在应用程序中使用框架，你需要约定你的应用程序如何与框架交互，例如，如果一个 Java 对象负责处理一个工具条按钮点击，框架需要知道调用这个对象的那个方法。

对于这种用途，每个框架定义一个应用程序编程接口（API），API 定义了你如何在你的代码中与框架交互，API 也定义了框架传教或控制的对象。通常对于这种用途，框架用继承或 annotation。

通过继承的 API 定义

定义 API 的“传统”方法是通过继承，这个方案需求你的类扩展或实现框架类和接口，Eclipse 3.x 平台 API 用这种方案。

框架定义了，例如，一个抽象的类，定义了要实现的方法，在工具条按钮例子中，方法可能叫做 `execute()`，框架知道在按钮点击时这个方法必须被调用。

通过继承的 API 定义是一个定义 API 的简单方案，但是它也紧紧的耦合类与框架，例如，离开框架去测试类是很困难大，它也让扩展或更新框架变的困难，因为更新可能会影响客户端，这是为什么 Eclipse 4.x 不再采用这种方案的原因。

通过 Annotation 的 API 定义

Eclipse 4.x 平台 API 基于 annotations，也就是说，annotations 用来标注在特定的点调用哪个方法，这些 annotations 是叫做行为 annotations。

下表列出了对于 Part 有效的行为 annotations.

Annotation	描述
<code>@PostConstruct</code>	在类构造以及字段和方法注入执行之后调用。
<code>@PreDestroy</code>	类销毁之前调用，可以用于清理资源
<code>@Focus</code>	在 Part 获得焦点时调用
<code>@Persist</code>	Eclipse 框架在 Part 上触发 Save 请求时调用
<code>@PersistState</code>	在模型对象 dispose 之前调用，这样 Part 有机会去保存它的实例状态，这个方法在 <code>@PreDestroy</code> 方法之前调用。

`@PostConstruct`, `@PreDestroy` annotations 包含在 `javax.annotation` 包中。`@Persist`, `@PersistState`, 以及 `@Focus` 是 `org.eclipse.e4.ui.di` 包的一部分。

Eclipse 为 commands 以及应用程序生命周期定义了附加的行为 annotations，在相应的章节中描述。

行为 annotations 隐含方法依赖注入

行为 annotations 隐含框架需要为方法提供指定的参数，也就是说，框架也执行方法依赖注入。如果你也加入了 `@Inject` annotation，方法将被调用 2 次，第一次是在依赖注入阶段，第二次是行为 annotation，这通常不是想要的结果，所以是一个错误。

用 `@PostConstruct` 方法来建立用户接口

推荐在 `@PostConstruct` annotation 的方法中构造 Part 的用户接口，也可以在构造器中构造用户接口，但是不推荐这么做，因为在这个点字段和方法注入还没有完成。

在 @PostConstruct 方法中建立用户接口需求 @Inject 方法明白用户接口还没有建立。

为什么 @PostConstruct 方法没有被调用？

Java 7 和 Eclipse 平台都输出了 @PostConstruct annotation，在你的 Eclipse 应用程序中，你需要告诉框架应该使用 Eclipse 平台的那个。

在你的 @PostConstruct 方法没有调用的情况下，确保你在 MANIFEST.MF 文件中定义了到 org.eclipse.core.runtime 的依赖。参考 <http://wiki.eclipse.org/Eclipse4/RCP/FAQ> 获得详细信息。

注意，org.eclipse.core.runtime 输出正确版本的 javax.annotation，如果你在 MANIFEST.MF 文件中依赖了 org.eclipse.core.runtime，就不需要额外的包了，如果，为了某些原因，你不希望依赖 org.eclipse.core.runtime，你可以定义一个包依赖 javax.annotation 包，并且设置版本为 1.0.0。

练习：用 @PostConstruct

实现 @PostConstruct 方法

在你的 `TodoOverviewPart`, `TodoDetailsPart` 以及 `PlaygroundPart` 类中添加下面的方法, 如果你为这些类建立了构造器, 你可以删除它们。

```
import javax.annotation.PostConstruct;
import org.eclipse.swt.widgets.Composite;

// more code

@PostConstruct
public void createControls(Composite parent) {
    System.out.println(this.getClass().getSimpleName()
        + " @PostConstruct method called.");
}
```

验证

运行你的应用程序, 并且验证 `@PostConstruct` 方法被调用了, 如果这没有工作的话, 参考前面的“为什么 `@PostConstruct` 方法没有调用?”。

菜单以及工具条应用程序对象

添加菜单和工具条

你可以通过应用程序模型添加菜单和工具条到你的 RCP 应用程序，这些项可以放在不同的位置，你可以，例如，添加菜单到窗口或 Part。

应用程序模型提供几个选项来贡献菜单和工具条项目，对于简单的情况，你可以用 Direct MenuItem 或者 Direct ToolItem 模型元素，它们包含一个引用到一个类，当相应的项选择时被执行，下面的描述称这些元素为: direct items。

如果你用 Handled MenuItem 以及 Handled ToolItem 模型元素，你引用到 Command 模型元素，Command 模型元素是在后面描述。

应用程序模型也支持在运行时创建菜单，通过 DynamicMenuContribution 模型元素。

应用程序的工具条通过 Trimbars 模型元素封装在应用程序模型中，一个 trimbar 可以为 TrimmedWindow 模型元素定义，通过他的 Side 属性，你可以定义它放在窗口的顶部，左边，右边，或者底部。

菜单和工具条支持分隔符，并且可以有子菜单。

Command 和 Handler 是什么？

Eclipse 应用程序模型允许你指定 commands 以及 handlers。

一个 command 是一个抽象的可以执行的动作的声明，例如，保存、编辑、或者拷贝。一个 command 与它的实现细节是分离的。

command 的行为通过 handler 定义，一个 handler 模型元素通过 handler 的 contributionURI 属性指向一个类，这个属性在模型编辑器中显示为 Class URI，本书中，这样的类叫做 handler 类。

Commands 被 Handled MenuItem 以及 Handled ToolItem 模型元素使用。

提示：优先使用 commands，而不是 direct items，用 commands 结合 handlers 允许你为不同的范围(应用程序或者 Part)定义不同的 handlers，并且你可以为 handlers 关联的 command 定义 key 绑定。

助记符

应用程序模型允许你定义助记符，一个助记符通过菜单中一个下划线字母表示，用户按下 ALT 键可以快速通过键盘访问菜单项。

你通过在字母前加上 & 符号来表述助记符，例如，&Save，当 ALT 键按下时，S 有下划线。

标准的 commands

Eclipse 4 没有提供标准的命令，也就是说，你必须在你的应用程序模型中建立所有需要的命令。

commands 和 handlers IDs 的命名方案

一个好的习惯是用你的项目的顶级包名作为 ID 的前缀，并且只用小写字母。

Commands 和 handlers 的 ID 应该表现它们的关系，例如，如果实现了一个 ID 为 com.example.contacts.commands.show 的 command，你应该为 handler 使用 ID com.example.contacts.handler.show。如果一个 command 有多个 handlers，在它们的 ID 后面添加后缀来描述它的用途，例如：com.example.contacts.handler.show.details。

在你视线公用功能时，例如，保存、拷贝，你应该用存在的平台 ID，某些 Eclipse 贡献期盼这些 ID 来更好的与 OS 整合（例如，在 Mac OS 中，Preferences 通常放在第一个菜单中），command ID 的完整列表在 `org.eclipse.ui.IWorkbenchCommandConstants` 中有效。

Command	ID
Save	<code>org.eclipse.ui.file.save</code>
Save All	<code>org.eclipse.ui.file.saveAll</code>
Undo	<code>org.eclipse.ui.edit.undo</code>
Redo	<code>org.eclipse.ui.edit.redo</code>
Cut	<code>org.eclipse.ui.edit.cut</code>
Copy	<code>org.eclipse.ui.edit.copy</code>
Paste	<code>org.eclipse.ui.edit.paste</code>
Delete	<code>org.eclipse.ui.edit.delete</code>
Import	<code>org.eclipse.ui.file.import</code>
Export	<code>org.eclipse.ui.file.export</code>
Select All	<code>org.eclipse.ui.edit.selectAll</code>
About	<code>org.eclipse.ui.help.aboutAction</code>
Preferences	<code>org.eclipse.ui.window.preferences</code>
Exit	<code>org.eclipse.ui.file.exit</code>

Handler 类的依赖注入

Handler 类和它们的行为 annotation

Direct menu, tool item 以及 handler 模型元素指向一个类，这个类用行为 annotation 来标注用户选择相关的用户接口元素时，框架将调用的方法。为了简洁，下面的描述用 handler 类来表示这样的类。

handler 类的行为 annotation 在下表描述：

Annotation	描述
@Execute	标注方法为 handler 类的 action，框架在相关的用户接口元素（例如菜单项）被选择的时候调用这个方法。
@CanExecute	标注方法可以被框架访问来检查 handler 类是否可以执行。如果 handler 类在这个方法中返回 false，Eclipse 禁止相应的用户接口元素。例如，当 handler 类在 @CanExecute 方法中返回 true 时，保存按钮是活动的。 这个方法默认是 true，意味着，如果 handler 类总是可以执行的话，它不需要实现一个 @CanExecute 方法。

警告：顺应 Javadoc，只有一个方法允许用 @Execute annotation，同样的规则对于 @CanExecute 一样，虽然框架当前对于几个方法用这些 annotation 标注不抱怨，但是你应该避免这样，否则哪个方法被调用是不确定的。

注意：Eclipse 运行时尝试注入这些方法中指定的所有参数。

下面的例子演示 handler 类的实现：

```
package com.example.e4.rcp.todo.handlers;

// import statements cut out
// ..

public class ExitHandler {
    @Execute
    public void execute(IWorkbench workbench) {
        workbench.close();
    }

    // NOT REQUIRED IN THIS EXAMPLE
    // just to demonstrates the usage of
    // the annotation
    @CanExecute
    public boolean canExecute() {
        return true;
    }
}
```

handler 类用哪个上下文？

handler 类在那个 handler 被调用的 IEclipseContext 中执行，也就是说，当前窗口中活动的那个，在大部分情况下，这是活动的 Part 的上下文。在你的应用程序启动时，handler 类初始化在另一个上下文，也就是应用程序或者窗口上下文。

所有需求的参数应该被注入到 @Execute 注解的方法，因为你希望 handler 类在执行时得到它的运行时信息。

警告：为了确保你从活动的上下文得到期盼的值注入到你的 handler 类，永远不要再它里面用字段或构造器注入。

Handlers 的范围

如果一个 command 被选择了，运行时将决定 command 对应的 handlers。

每个 command 在一个给定的范围只有一个有效的 handler，应用程序模型允许你为应用程序、一个窗口、或者一个 Part 创建一个 handler。

如果为 command 指定了多个 handler，Eclipse 将选择对于模型元素最特殊的那个 handler。

例如，假设对于 copy command 有 2 个 handler，一个是 for 窗口，一个是 for Part，那么运行时选择接近当前用户选择的模型元素的那个。

当 handler 选择后，@CanExecute 被调用，因此 handler 可以决定是否有能力在给定的上下文中执行，如果它返回 false，那么将禁止指向那个 command 的菜单和工具项。

@CanExecute 的执行

如果在上下文中发生了改变，通过 @CanExecute 注解的方法将被框架调用，也就是说，如果上下文被修改了，或者活动的上下文改变了，应用程序代码可以通过事件代理(broker)发送事件来请求框架执行 @CanExecute 方法。

```
// evaluate all @CanExecute methods
eventBroker.send(UIEvents.REQUEST_ENABLEMENT_UPDATE_TOPIC, UIEvents.ALL_ELEMENT_ID);

// evaluate a context via a selector
Selector s = (a selector that an MApplicationElement or an ID);
eventBroker.send(UIEvents.REQUEST_ENABLEMENT_UPDATE_TOPIC, s);

//See https://bugs.eclipse.org/bugs/show\_bug.cgi?id=427465 for details
```

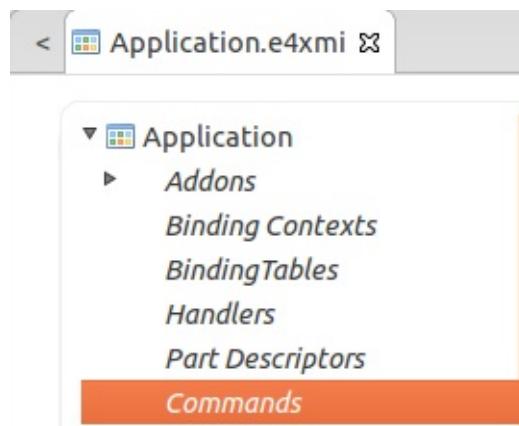
练习：添加菜单

本练习的目标

在本练习中，你将为你的应用程序建立 commands 以及 handlers，然后，你将建立菜单项来使用这些 commands。

建立 command 模型元素

打开 com.example.e4.rcp.todo 插件的 Application.e4xmi 文件，然后选择 Commands 项，这个选择在下图中高亮显示：



通过 Add... 按钮，你可以创建新的 commands，名称和 ID 是重要的字段，创建下列的 commands。

ID	名称
org.eclipse.ui.file.saveAll	Save
org.eclipse.ui.file.exit	Exit
com.example.e4.rcp.todo.command.new	New Todo
com.example.e4.rcp.todo.command.remove	Remove Todo
com.example.e4.rcp.todo.command.test	For testing

建立 handler 类

为你的 handler 类创建 com.example.e4.rcp.todo.handlers 包。

所有的 handler 类实现通过 @Execute 注解的 execute() 方法。

```
package com.example.e4.rcp.todo.handlers;

import org.eclipse.e4.core.di.annotations.Execute;

public class SaveAllHandler {
    @Execute
    public void execute() {
        System.out.println((this.getClass().getSimpleName() + " called"));
    }
}
```

为所有的类使用这个模板，实现下列的类。

- SaveAllHandler
- ExitHandler
- NewTodoHandler
- RemoveTodoHandler
- TestHandler

建立 Handler 模型元素

在你的应用程序模型中选择应用程序范围 Handlers 项，然后为你的 commands 创建下表中的 handlers。对于 handler 的定义，ID、command、以及 class 是相关的信息。

为所有 handlers 使用 com.example.e4.rcp.todo.handler 前缀。

Handler ID	Command	Class
.saveall	Save	SaveAllHandler
.exit	Exit	ExitHandler
.new	New Todo	NewTodoHandler
.remove	Remove Todo	RemoveTodoHandler
.test	For testing	TestHandler

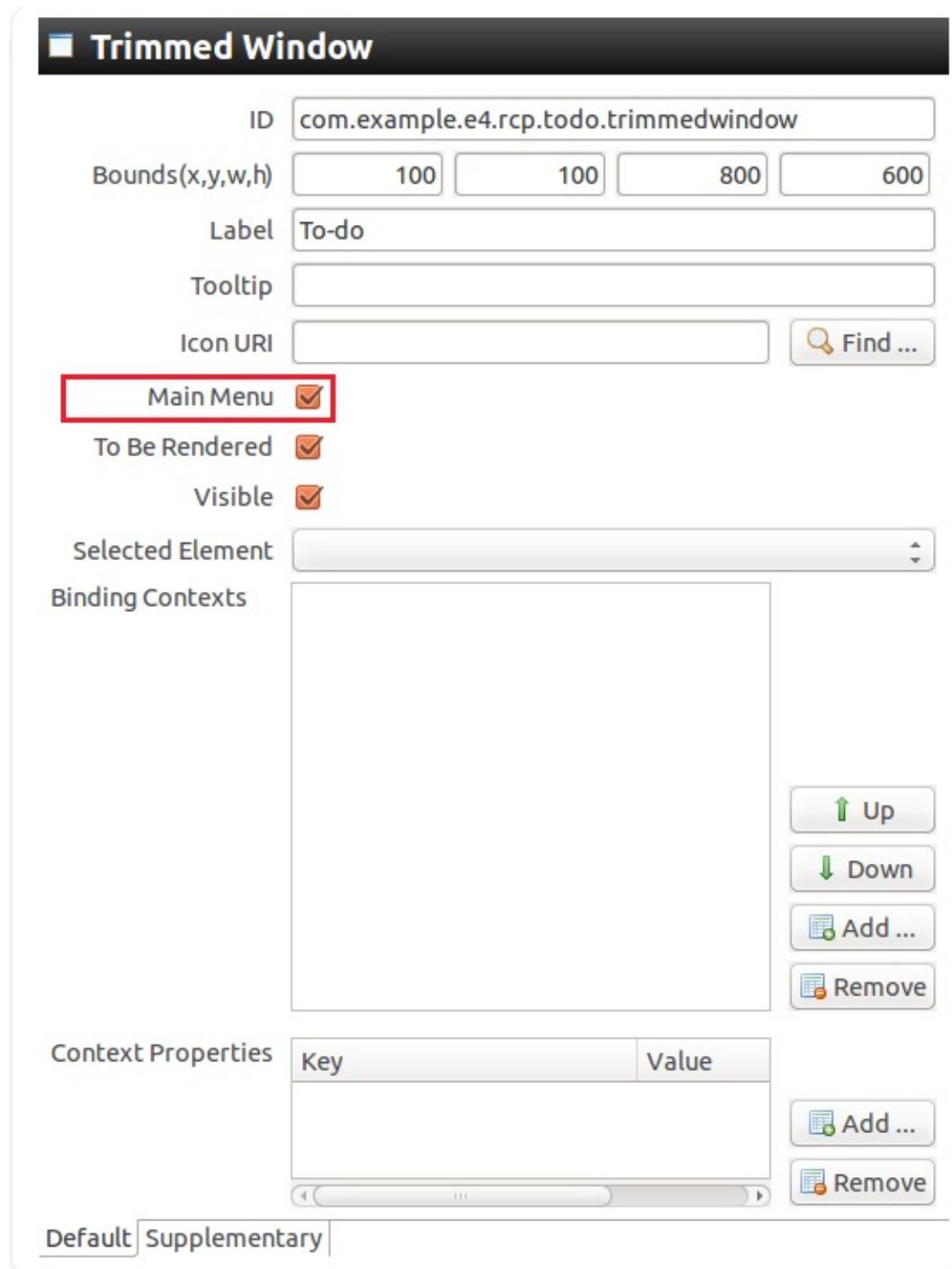
应用程序模型编辑器显示了 command 的名称和 ID，类 URI 跟随 bundleclass:// 方案，表中仅定义了类名是为了让表格更容易阅读。例如，对于 save handler，看起来如下：

```
bundleclass://com.example.e4.rcp.todo/com.example.e4.rcp.todo.handlers.SaveAllHandler
```



添加一个菜单

在 Application.e4xmi 文件中选择 TrimmedWindow 项，然后勾选 Main Menu 属性。

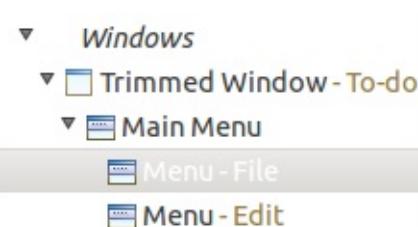


为主菜单指派 ID `org.eclipse.ui.main.menu`。

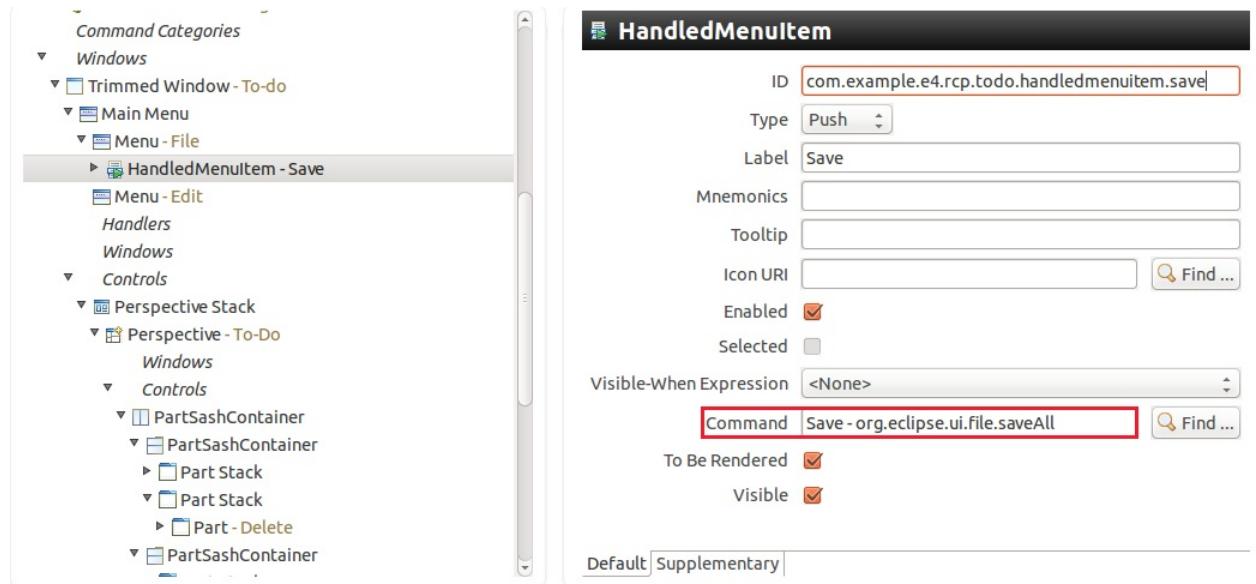
警告：确保主菜单的 ID 是正确的，你将在后面通过其它插件用它来贡献菜单。

添加 2 个菜单，一个名称为 `File`，另一个名称为 `Edit`。

为 `File` 菜单设置 ID 为 `org.eclipse.ui.file.menu`，为 `Edit` 菜单设置 ID 为 `com.example.e4.rcp.todo.menu.edit`。



添加一个 Handled MenuItem 模型元素到 File 菜单，这个项应该通过 Command 属性指向 Sava command。



在 Save 菜单之后添加一个分隔符，再之后为 Exit command 添加一项。

添加所有其他 commands 到 Edit 菜单。

为 Exit 实现一个 handler

为测试你的 handler 是否可以工作，改变你的 ExitHandler 类，当选择的时候，关闭应用程序。

```
package com.example.e4.rcp.todo.handlers;

import org.eclipse.e4.core.di.annotations.Execute;
import org.eclipse.e4.ui.workbench.IWorkbench;

public class ExitHandler {
    @Execute
    public void execute(IWorkbench workbench) {
        workbench.close();
    }
}
```

验证

验证当你选择 Save 菜单时，save handler 得到调用。

也要检查你能够通过 Exit 菜单项推出应用程序。

可能的问题：Exit 菜单项 on Mac OS

If you use the "org.eclipse.ui.file.exit" ID for your exit command, the Eclipse framework tries to map the exit command to its default menu location on the MacOS. If you don't see your exit menu, in its defined position, check this location.

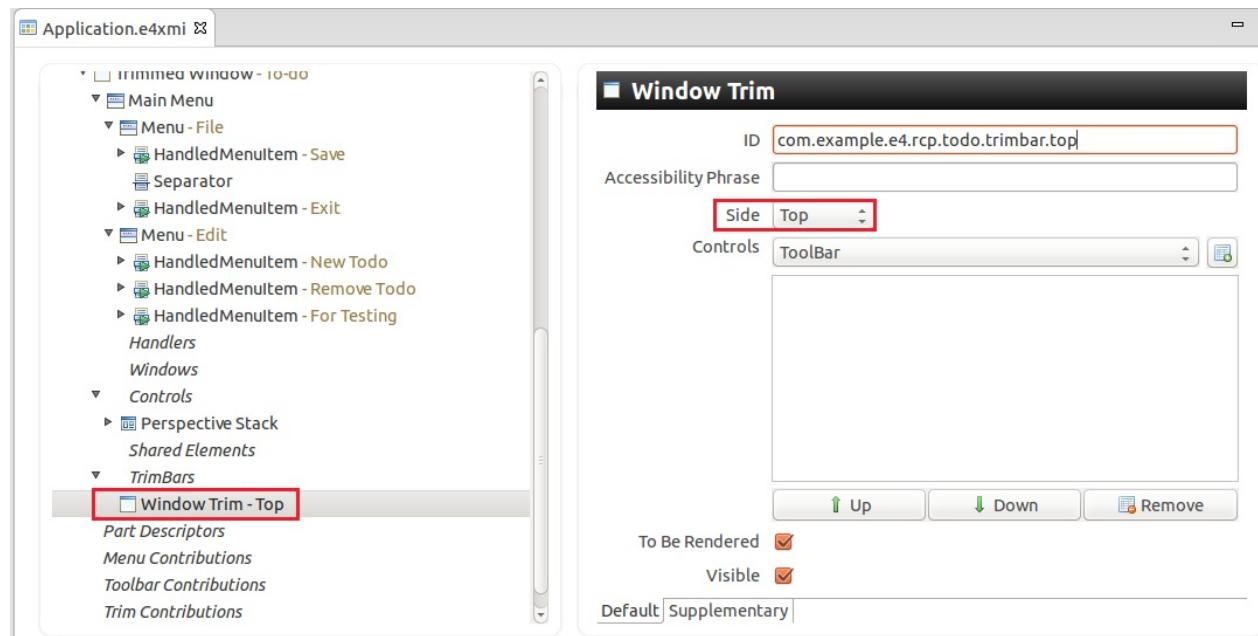
练习：添加一个工具栏

本练习的目标

在本练习中，你将添加工具栏到应用程序。

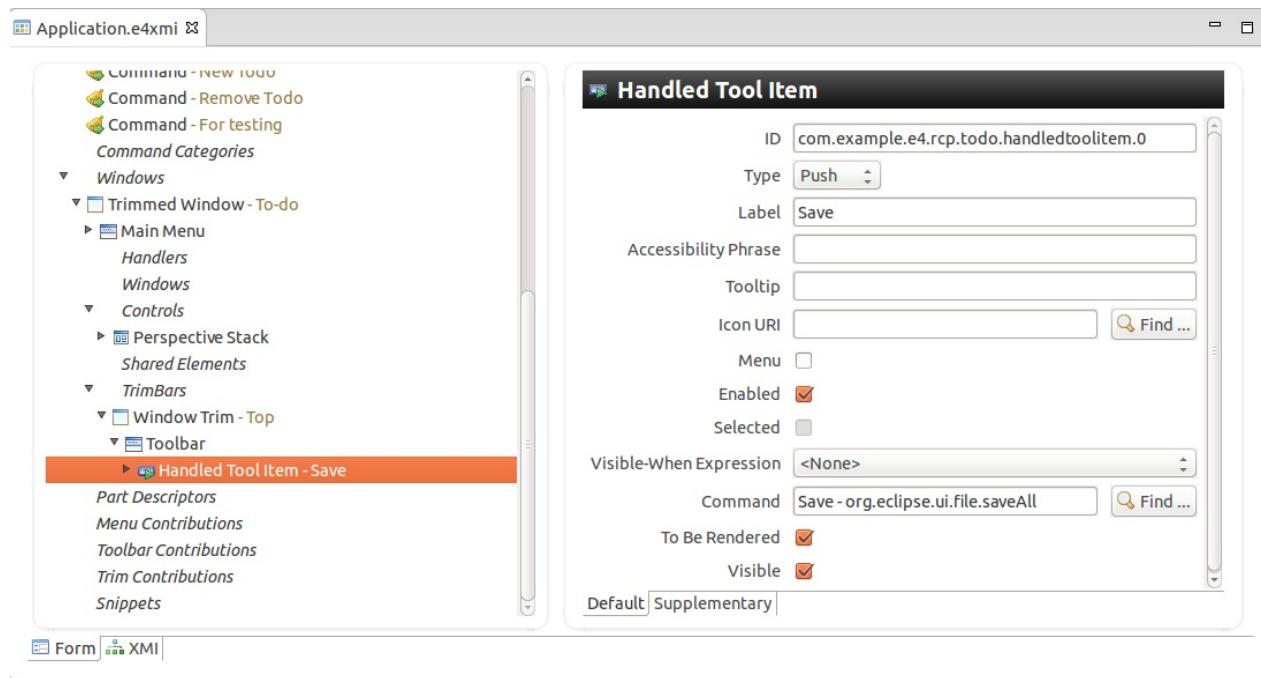
添加一个工具栏

在 TrimmedWindow 项下面选择 TrimBars，然后点击 添加... 按钮，Side 属性应该置顶，所有指派给那个 trimbar 的其它工具栏出现在应用程序的顶端。



添加一个 ToolBar 模型元素到 TrimBar，添加一个 Handled ToolItem 到这个工具栏，指向 `org.eclipse.ui.file.saveAll` command。

将这项的 Label 设置为 Save。



验证

验证当你从菜单或者工具栏选择 Save 时，你的 save handler 被调用了。

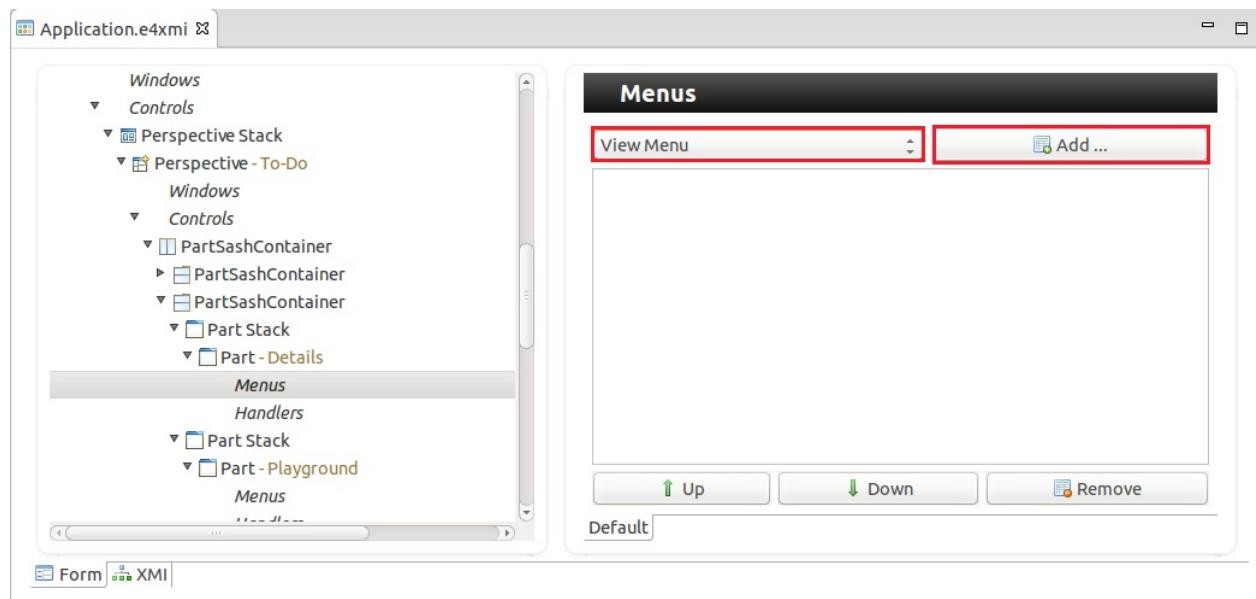
视图、弹出以及动态菜单

视图菜单

Part 中的一个菜单可以定义为 View 菜单。

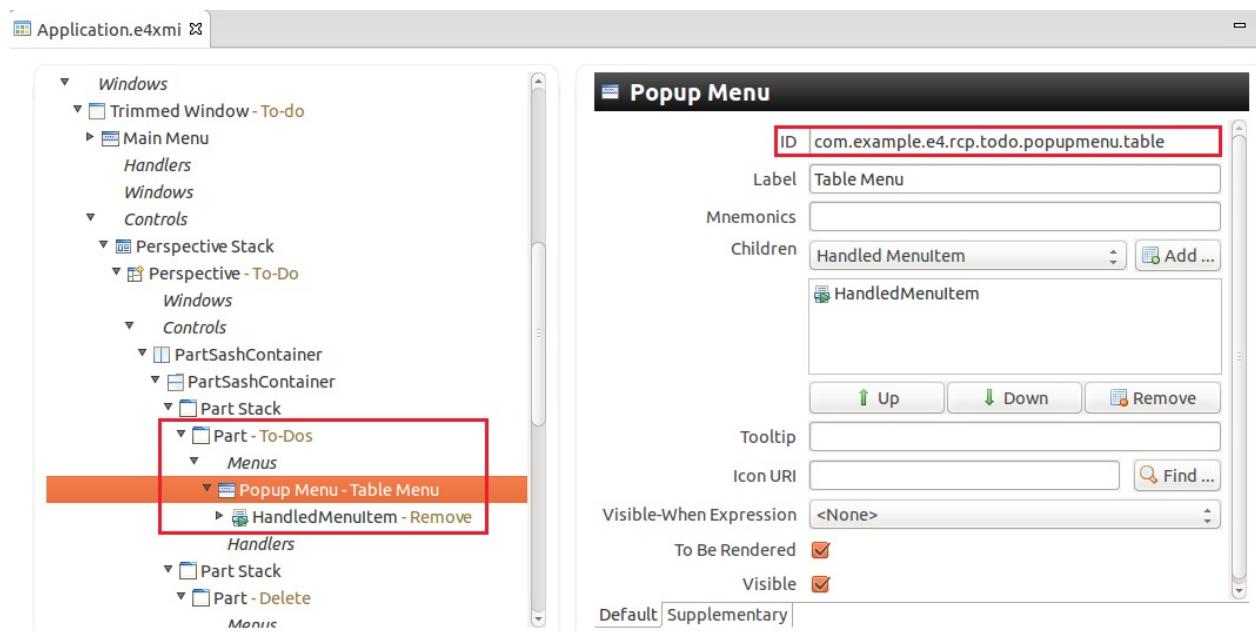
注意：默认的 Eclipse（渲染）框架对于一个 Part 只支持一个菜单。

要添加 View 菜单项，在 Part 下方选择菜单，然后添加一个 ViewMenu 模型项到它。

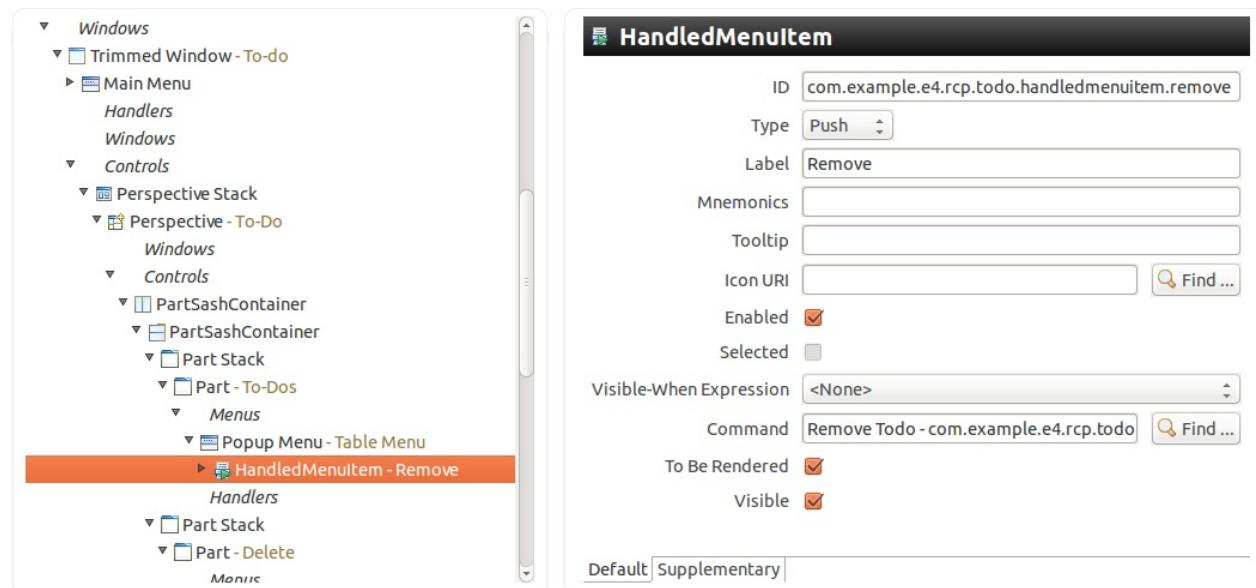


弹出菜单

你也可以通过应用程序模型为 SWT 控件定义弹出菜单，为包含 SWT 控件的 Part 创建一个弹出菜单。



弹出菜单包含项，例如，一个 HandledMenuItem。



之后，弹出菜单可以指派给一个带有 EMenuService 服务的 SWT 控件，可以通过依赖注入来访问。这个类提供了 registerContextMenu(control, id) 方法来作此用途。registerContextMenu 方法的参数 id 必须是你的弹出菜单模型元素的 ID 属性。

下面的伪代码演示了一个注册的例子，它用 JFace viewer，因为弹出菜单需要注册到 SWT 控件，例子代码演示了如何访问这个控件。

```
package com.example.e4.rcp.todo.parts;

import javax.annotation.PostConstruct;

import org.eclipse.e4.ui.services.EMenuService;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;

public class TodoOverviewPart {

    @PostConstruct
    public void createControls(Composite parent, EMenuService menuService) {
        // more code...
        TableViewer viewer = new TableViewer(parent, SWT.FULLSELECTION | SWT.MULTI);

        // more code

        // register context menu on the table
        menuService.registerContextMenu(viewer.getControl(),
            "com.example.e4.rcp.todo.popupmenu.table");
    }
}
```

如果你希望实现这个例子，在插件的 MANIFEST.MF 文件中必须定义依赖于 `org.eclipse.e4.ui.workbench.swt`，`org.eclipse.e4.ui.services`，`org.eclipse.e4.ui.model.workbench` 插件。

动态菜单和工具栏项

你也可以通过 `DynamicMenuContribution` 模型元素在运行时创建菜单和工具条。

This model element points to a class in which you annotate a method with the annotation. The annotated method is called if the user selects the user interface element.

The `@AboutToHide` annotation can be used to annotate a method which is called before the menu is hidden.

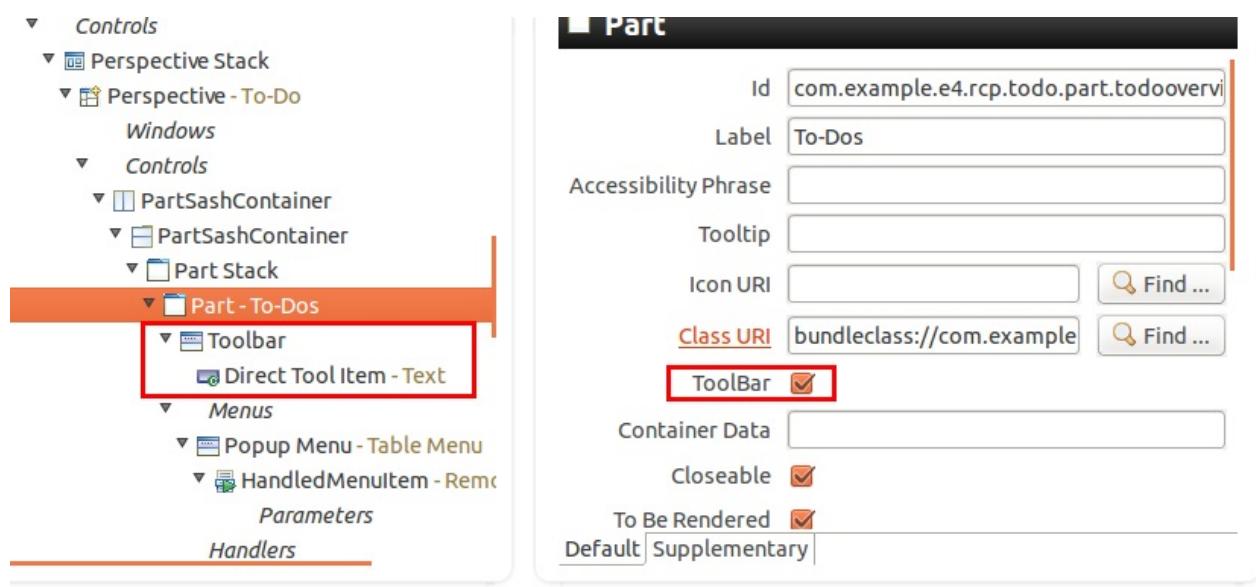
In these methods you can dynamically adjust the application model.

工具栏、工具控制以及下拉工具项

添加工具栏到 Parts

要在 view 中添加工具条，在 part 的模型元素中设置 Toolbar 标志，然后在应用程序模型中创建条目。

这样的一个例子在下面的截图中显示：



ToolControls

ToolControl 模型元素指向可以创建在工具栏中显示的控制的 Java 类。

例如，下面的代码在工具栏中创建一个 Text 字段，看起来像一个查询字段。

```
package com.example.e4.rcp.todo;

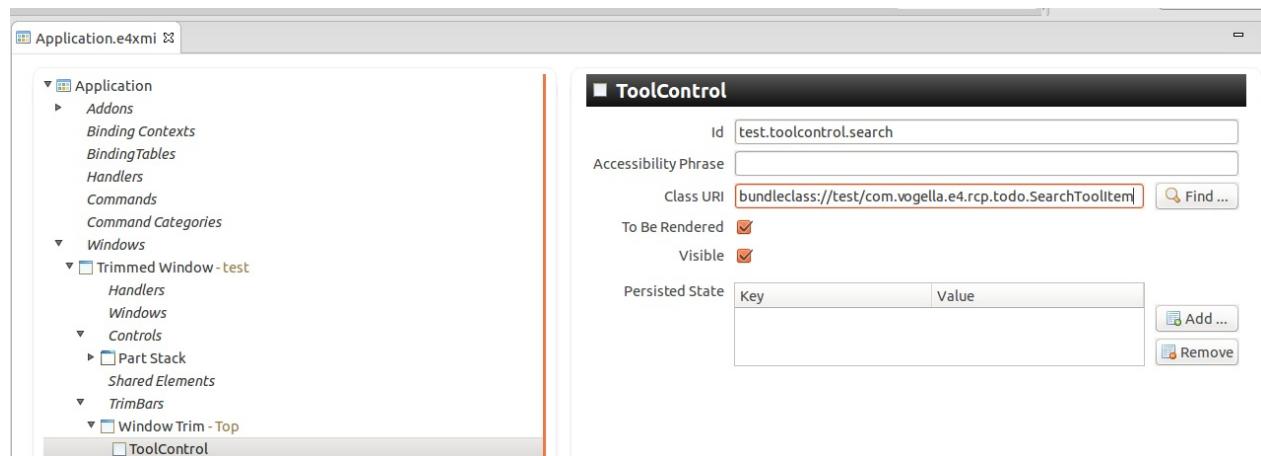
import javax.annotation.PostConstruct;

import org.eclipse.jface.layout.GridDataFactory;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Text;

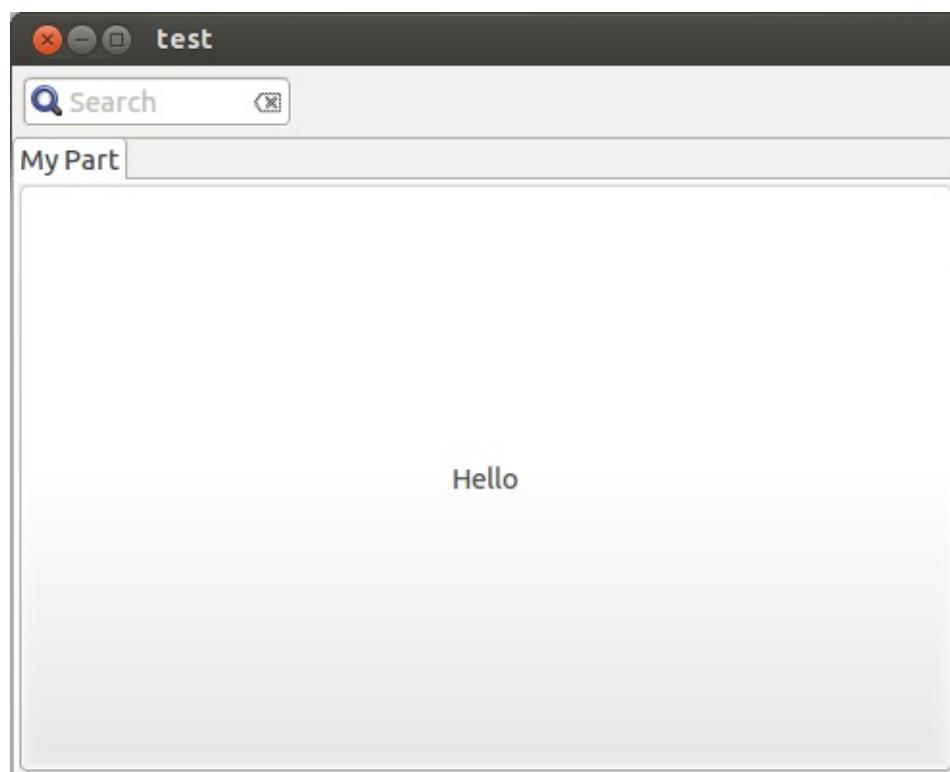
public class SearchToolItem {
    @PostConstruct
    public void createControls(Composite parent) {
        final Composite comp = new Composite(parent, SWT.NONE);
        comp.setLayout(new GridLayout());
        Text text = new Text(comp, SWT.SEARCH | SWT.ICON_SEARCH | SWT.CANCEL
            | SWT.BORDER);
        text.setMessage("Search");
        GridDataFactory.fillDefaults().hint(130, SWT.DEFAULT).applyTo(text);

    }
}
```

你也可以加入这样的 ToolControl，例如，到你的 trimbar，在下图中解释：

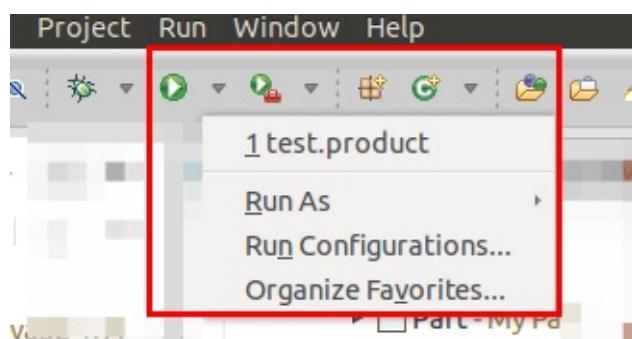


下面的截图显示这个 ToolControl 用在例子 RCP 应用程序中。



下拉工具项

在一个 toolitem 上设置 Menu 属性，就可以定义一个类似于 Eclipse IDE 中 Run As ... 按钮的菜单，在下图中解释：

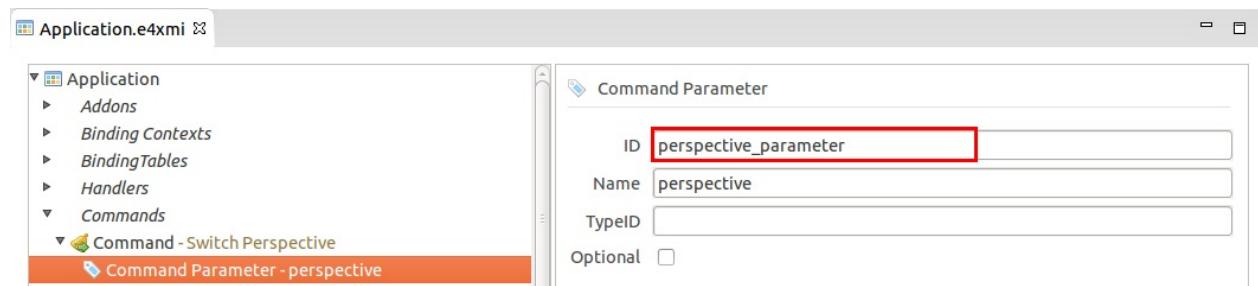


更多关于 commands 和 handlers

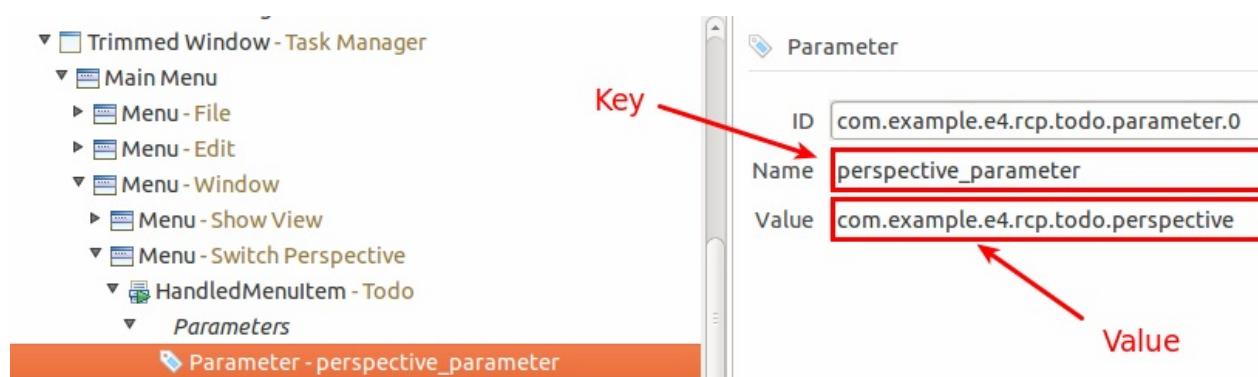
传递参数到 commands

你也可以传递参数到 commands，选择一个 command，然后在参数下点击[添加]按钮。

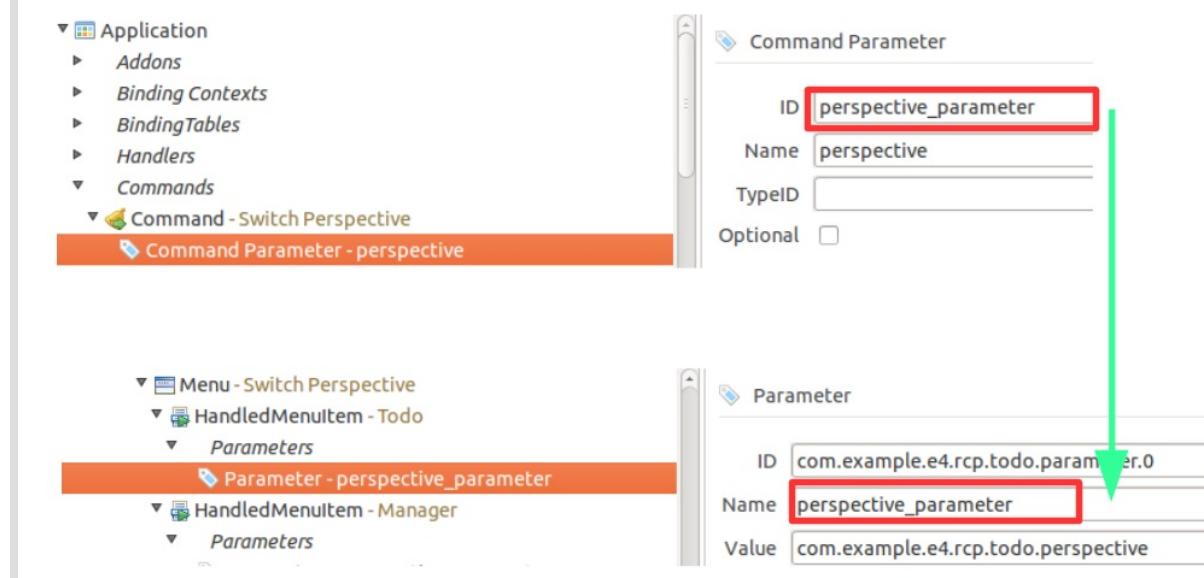
ID 是你用来获取通过 @Named 注解注入的参数值的标识，



在你的 HandledMenuItem 或 HandledToolItem 中，添加一个参数，然后将来自 Name 字段定义的 command 参数放进去。Value 字段传递给 command 的 handler。



警告，参数的 ID 是重要的，ID 必须通过 @Named 注解来注入，并且在定义菜单或工具条时用作 Name(第二个字段)。这在下面的图中高亮显示：



为了得到注入到你的 handler 类中的参数，你通过 @Named 注解指定参数的 ID，这通过下面的代码演示：

```

package com.example.e4.rcp.todo.handlers;

import java.util.List;

import javax.inject.Named;

import org.eclipse.e4.core.di.annotations.Execute;
import org.eclipse.e4.ui.model.application.MApplication;
import org.eclipse.e4.ui.model.application.ui.advanced.MPerspective;
import org.eclipse.e4.ui.model.application.ui.basic.Mwindow;
import org.eclipse.e4.ui.workbench.modeling.EModelService;
import org.eclipse.e4.ui.workbench.modeling.EPartService;

// switcher perspectives based on a command parameter
public class PerspectiveSwitchHandler {

    @Execute
    public void switchPerspective(Mwindow window,
        EPartService partService,
        EModelService modelService,
        @Named("perspective_parameter") String perspectiveId) {
        // use parameter to find perspectives
        List<MPerspective> perspectives = modelService.findElements(window,
            perspectiveId, MPerspective.class, null);

        // switch to perspective with the ID if found
        if (!perspectives.isEmpty()) {
            partService.switchPerspective(perspectives.get(0));
        }
    }
}

```

提示：除了注入每个参数，你也可以注入 ParameterizedCommand command，然后通过 API 来访问参数。

```

package com.example.e4.rcp.todo.handlers;

import javax.inject.Named;

import org.eclipse.e4.core.di.annotations.Execute;

public class TestHandlerWithCommandInjected {
    private String parametername = "com.example.e4.rcp.todo" +
        ".commandparameter.input";
    @Execute
    public void execute(ParameterizedCommand command) {
        Object queryId = command.getParameterMap().get(parametername);
        // more...
    }
}

```

Usage of core expressions

The visibility of menus, toolbars and their entries can be restricted via core expressions. You add the corresponding attribute in the application model to the ID defined by the org.eclipse.core.expressions.definitions extension point in the plugin.xml file.

To add this extension point to your application, open the plugin.xml file and select the Dependencies tab in the editor. Add the org.eclipse.core.expressions plug-in in the Required Plug-ins section.

Afterwards select the Extensions tab, press the Add button and add the org.eclipse.core.expressions.definitions extension. You define an ID under which the core expression can be referred to in the application model.

Via right-click on the extension you can start building your expression.

The following example can be used to restrict the visibility of a menu entry based on the type of the current selection. You will later learn how to set the current selection. Please note that the variable for the selection is currently called org.eclipse.ui.selection. In Eclipse 3.x this variable is called selection.

```
<extension
    point="org.eclipse.core.expressions.definitions">
<definition
    id="com.example.e4.rcp.todo.selectionset">
<with variable="org.eclipse.ui.selection">
<iterate ifEmpty="false" operator="or">
    <instanceof value="com.example.e4.rcp.todo.model.Todo">
    </instanceof>
</iterate>
</with>
</definition>
</extension>
```

You can assign this core expression to your menu entry in the application model. It can be used to restrict the visibility of model elements.

HandledMenuItem

Id: com.example.e4.rcp.todo.menusitems.test

Type: Push

Label: Test

Mnemonics:

Tooltip:

Icon URI:

Enabled:

Selected:

Visible-When Expression: CoreExpression

Command: For testing - com.example.e4.rcp.todo.test

To Be Rendered:

Visible:

Default **Supplementary**

Part Descriptors

- ▶ Commands
- Command Categories
- ▼ Windows
 - Trimmed Window - To-do
 - >Main Menu
 - ▶ File
 - ▶ Edit
 - ▶ HandledMenuItem - New Todo
 - ▶ HandledMenuItem - Remove Todo
 - ▶ HandledMenuItem - Test

Core Expression

Expression Id: com.example.e4.rcp.todo.selectionset

This approach is similar to the definition of core expressions in Eclipse 3.x.

The values available for Eclipse 3.x are contained in the ISources interface and documented in the Eclipse core expressions wiki . Eclipse 4 does not always support the same variables, but the wiki documentation might still be helpful.

Evaluate your own values in core expressions

You can also place values in the IEclipseContext of your application and use these for your visible-when evaluation.

The following code demonstrates an example handler class which places a value for the myactivePartId key in the context (you will learn more about modifying the IEclipseContext later).

```
@Execute
public void execute(IEclipseContext context) {
    // put an example value in the context
    context.set("myactivePartId",
    "com.example.e4.rcp.todo.part.todooverview");
}
```

The following shows an example core expression which evaluates to true if an `myactivePartId` key with the value `com.example.e4.rcp.ui.parts.todooverview` is found in the context.

```
<extension
    point="org.eclipse.core.expressions.definitions">
<definition
    id="com.example.e4.rcp.todo.todooverviewselected">
<with
    variable="myactivePartId">
<equals
    value="com.example.e4.rcp.todo.part.todooverview">
</equals>
</with>
</definition>
</extension>
```

This core expression can get assigned to a menu entry and control the visibility.



Key bindings

Using key bindings in your application

It is also possible to define key bindings (shortcuts) for your Eclipse application. This requires two steps, first you need to enter values for the Binding Context node of your application model.

Afterwards you need to enter the key bindings for the relevant binding context in the BindingTable node of your application model. A binding table is always assigned to a specific binding context. A binding context can have several binding tables assigned to it.

Binding contexts are defined hierarchically, so that key bindings in a child override the matching key binding in the parent.

Warning Even though they sound similar a binding context is used for keybindings while the Eclipse context (IEclipseContext) is used as source for dependency injection.

JFace default values for binding contexts

The binding context is identified via its ID. They can get assigned to a window or a part in the application model. This defines which keyboard shortcuts are valid for the window and which are valid for the part.

Eclipse JFace uses predefined IDs to identify binding contexts. These IDs are based on the org.eclipse.jface.contexts.IContextIds class. JFace distinguishes between shortcuts for dialogs, windows or both.

The following table gives an overview of the supported IDs and their validity.

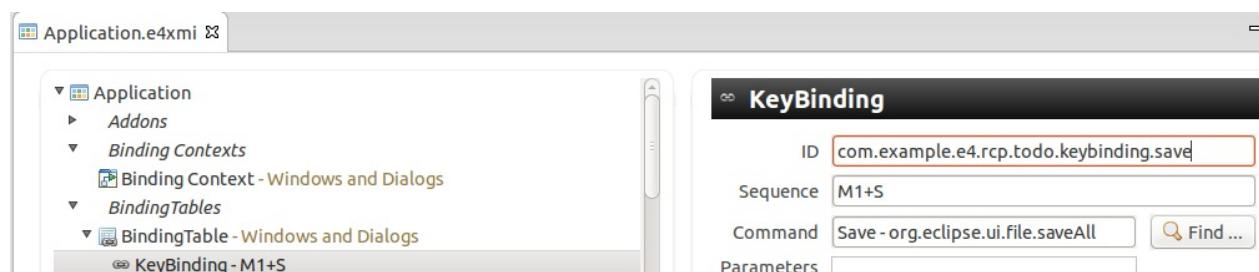
Context ID	Description
org.eclipse.ui.contexts.dialogAndWindow	Key bindings valid for dialogs and windows
org.eclipse.ui.contexts.dialog	Key bindings valid for dialogs
org.eclipse.ui.contexts.window	Key bindings valid for windows

As an example, Ctrl+C (Copy) would be defined in dialogAndWindows as it is valid everywhere, but F5 (Refresh) might only be defined for a Window and not for a Dialog.

Define Shortcuts

The BindingTable node in the application model allows you to create key bindings based on a binding context. For this you create a new BindingTable model element and define a reference to the binding context via its ID.

In your key binding entry you specify the key sequence and the command associated with this shortcut.



The control keys are different for each platform, e.g., on the Mac vs. a Linux system. You can use Ctrl, but this would be hardcoded. It is better to use the M1 - M4 meta keys.

Control Key	Mapping for Windows and Linux	Mapping for Mac
M1	Ctrl	Command
M2	Shift	Shift
M3	Alt	Alt
M4	Undefined	Ctrl

These values are defined in the `SWTKeyLookup` class.

Activate bindings

If there are several valid key bindings defined, the `ContextSet` class is responsible for activating one of them by default. `ContextSet` uses the binding context hierarchy to determine the lookup order. A binding context is more specific depending on how many ancestors are between it and a root binding context (the number of levels it has). The most specific binding context is considered first, the root one is considered last.

You can also use the `EContextService` service which allows you to explicitly activate and deactivate a binding context via the `activateContext()` and `deactivateContext()` methods.

Key bindings for a part

You can assign a specific binding context to be active while a part is active.

Part

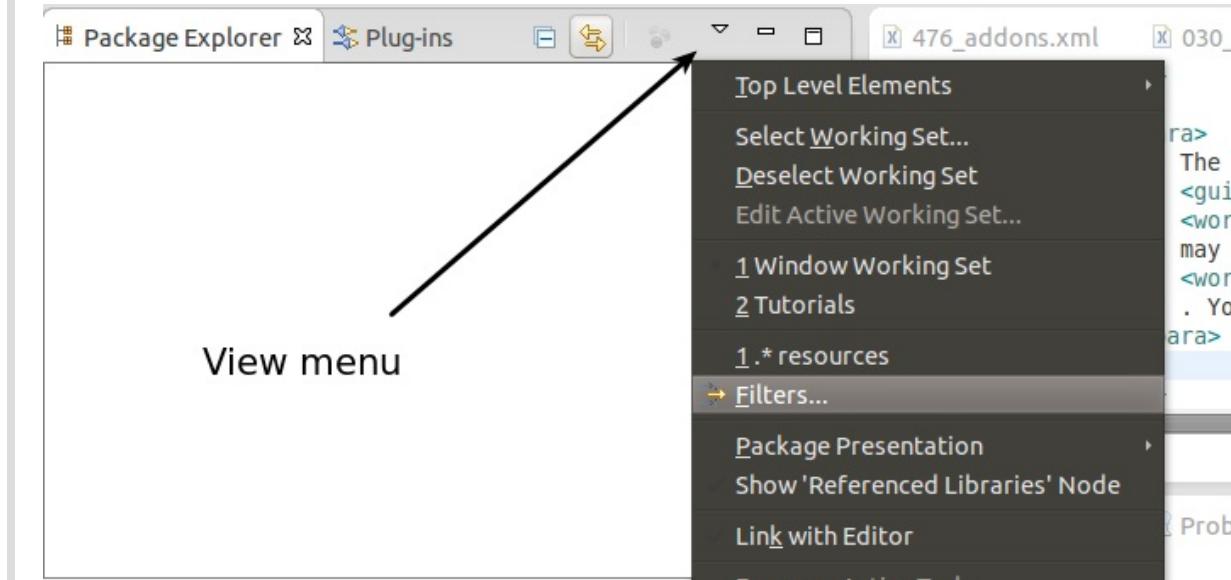
Id	<input type="text"/>				
Label	<input type="text"/> Todo Overview				
Accessibility Phrase	<input type="text"/>				
Tooltip	<input type="text"/>				
Icon URI	<input type="text"/> Find ...				
Class URI	<input type="text"/> bundleclass://com.example.e4.rcp.todo/com.e Find ...				
ToolBar	<input type="checkbox"/>				
Container Data	<input type="text"/>				
Closeable	<input type="checkbox"/>				
To Be Rendered	<input checked="" type="checkbox"/>				
Visible	<input checked="" type="checkbox"/>				
Binding Contexts	Binding Context - test				
	Up Down Add ... Remove				
Persisted State	<table border="1"><thead><tr><th>Key</th><th>Value</th></tr></thead><tbody><tr><td>Default</td><td>Supplementary</td></tr></tbody></table>	Key	Value	Default	Supplementary
Key	Value				
Default	Supplementary				

Key bindings assigned to a part are valid in addition to the key bindings provided by the currently active binding context, i.e. your global key bindings are still active in addition with the key bindings of the part.

Enable to start your product with right mouse click

You can also add the pde nature to your project in which you placed the product configuration file, if you want to be able to start your product via a right-click on the product and by selecting Run-as → Eclipse Application.

Note The Package Explorer view may have a filter set for .*resources. You can modify this filter via the view menu as depicted in the following screenshot.



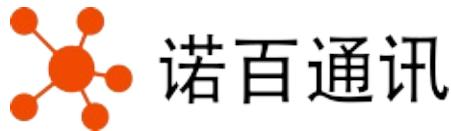
For this purpose remove the filter in the Package Explorer view for files starting with . (dot) and modify the .project file to the following.

```
<?xml version="1.0" encoding="UTF-8"?>

<projectDescription>
    <name>com.example.e4.rcp.todo.product</name>
    <comment></comment>
    <projects>
    </projects>
    <buildSpec>
        <buildCommand>
            <name>org.eclipse.pde.ManifestBuilder</name>
            <arguments>
            </arguments>
        </buildCommand>
        <buildCommand>
            <name>org.eclipse.pde.SchemaBuilder</name>
            <arguments>
            </arguments>
        </buildCommand>
    </buildSpec>
    <natures>
        <nature>org.eclipse.pde.PluginNature</nature>
    </natures>
</projectDescription>
```

中文翻译

By



此翻译仅用于学习，作者不保证翻译的正确性，版权归原始作者所有。

英文原文地址：<http://www.vogella.com/tutorials/EclipseRCP/article.html>