
学校的理想装备
电子图书·学校专集
校园网上的最佳资源
Visual j++6.0 基础
BOOK
内部资料 非卖品

返回总目录

# 第一部分

# Visual J++ 6.0 基础

# 目　　录

# 第 1 章　创建项目

项目是组成应用程序的文件集合。在 Visual J++中，项目是基于目录的，这也就意味着项目是由硬盘上的项目目录结构中的文件来定义的。

Visual J++提供了几种代码模板用来帮助用户创建项目。模板是框架式的 Java 类，它提供基本的代码框架。使用这些模板，可以创建下列项目类型：

- Windows 应用程序：使用 Windows Foundation Classes for Java（WFC，Windows 基础类），可以写出具有所有 Windows 特性的应用程序。Java 类扩展 com.ms.wfc.ui.From，并且可以使用 Forms Designer（窗体设计器）的 RAD 属性来修改窗体。Visual J++还提供 Application Wizard（应用程序向导）来创建 Windows 应用程序。
- 动态 HTML 应用程序：使用 WFC 类在 Dynamic HTML（动态 HTML）中编程。Java 类扩展 com.ms.wfc.html.Dhdocument。
- COM DLL：可以创建一个 Java 类，将它封装到一个 COM DLL 中，并且在任何支持 COM 的应用程序中使用它。
- 控件：用户可以使用 WFC 创建自己的控件。Java 类扩展 com.ms.wfc.ui.User-Control，并且能够在 Forms Designer 中修改。
- Java 控制台应用程序：控制台应用程序具有非图形方式的用户界面，

可以使用 WFC 和 Java API 中的非图形类来开发程序。

- Java 小应用程序：小应用程序是在 HTML 页中运行的应用程序。Java 类扩展 java.applet.Applet，并且主要使用在 java.awt 包中定义的类。

用户可以在不使用任何代码模板的情况下创建一个空的项目。

Visual J++项目与一个.vjp 文件关联，它跟踪某个项目设置，然后每一个项目包含在一种解决方案中。一种解决方案可以包含单一项目或多个项目，并且由一个.sln 文件来标识它们。有关如何添加多个项目到一个解决方案中的信息，见本章稍后一些的“创建多项目解决方案”一节。

除了创建新项目外，还可以：

- 使用 Project Explorer（项目管理器）管理项目。
- 设置项目选项。
- 从 Visual J++ 1.1 中导入一个项目。

## 基于目录的项目

在 Visual J++中，项目是基于目录的，这也就意味着项目是由硬盘上项目目录结构中的文件来定义的。例如：

- 添加文件或文件夹到项目中也就是将该项添加到在硬盘上的项目目录结构中。
- 通过文件系统添加文件到项目目录结构中也就是将该文件添加到该项目中，所提供的文件类型在项目过滤器（project filter）中指定。Visual

J++使用过滤器来确定哪类文件属于哪个 Java 项目。更多信息见本章稍后的“项目过滤器”一节。

- 通过文件系统添加文件夹到项目目录结构中也就是添加该文件夹到该项目中。在该文件夹中，所有在项目过滤器中指定文件类型的文件都添加到该项目中。
- 移动或复制文件或文件夹到项目中，也就是移动或复制该项目到硬盘上，反之亦然。
- 在项目中重命名文件或文件夹，则重命名在硬盘上的该项，反之亦然。

**注意：**当从项目中删除文件或文件夹时，可以选择在硬盘上保留该项或是选择完全删除它。

有关基于目录的项目的更多信息，见在 Visual Studio 在线文档中的“Different Project Models in Source Code Control”。

除了项目结构和文件系统之间的关系外，项目中 Java 软件包的分层结构应直接映射到在该项目（或文件系统）中的文件夹分层结构。在 Project Explorer 中，用户可以选择以下列两种视图之一来显示项目：

- 目录视图（默认的视图）显示一个所有项目子文件夹的分层结构列表，该结构与在硬盘上的项目目录结构一致。在项目中的每个 Java 软件包作为一个子文件夹显示。
- 软件包视图将项目的子文件夹作为 Java 软件包显示。注意，软件包视图只显示.java 文件和包含.java 文件的文件夹。每个显示的文件夹按照它的全限定名称列出。该项目节点本身认为是默认的软件包。

有关如何在 Project Explorer 中设置视图的更多信息，见本章稍后的“选择项目视图”一节。

## 使用 WFC 创建 Windows 应用程序

使用 WFC，用户可以使用 Java 程序语言创建 Windows 应用程序。还可以使用 Form Designer（窗体设计器）中的 RAD 特性在窗体上快速放入控件、配置它们的属性并且添加事件处理程序。

**注意：**在使用下面列出的过程创建 Windows 应用程序之前，关闭所有打开的项目[在 File(文件)菜单中，单击 Close All(全部关闭)]。

**使用 WFC 创建 Windows 应用程序**

1. 在 File 菜单上，单击 New Project（新项目）。
2. 在 New（新建）选项卡上，展开 Visual J++ Projects 文件夹，并且单击 Applications。然后选择 Windows Application 图标。
3. 在 Name（名称）框中，输入要创建的项目名称。
4. 在 Location（位置）框中，输入要保存该项目的路径，或单击 Browse（浏览）按钮来定位到某个文件夹。
5. 单击 Open（打开）。一个项目的折叠视图出现在 Project Explorer。
6. 在 Project Explorer 中，扩展该项目节点。一个带有默认文件名 Form1.Java 的文件已经添加到该项目上。

**注意：**在 Project Explorer 中改变这些文件名不会改变在源代码中与之相关联的类，反之亦然。用户必须手工改变旧名字的所有实例（注意，可以创建一个空项目，并且使用 Form（窗体）模板添加一个窗体。利用这个两步的过程，可以在窗体创建之前命名该窗体）。

当使用 Windows Application 模板时，Java 类展开 com.ms.wfc.ui.Form。要在 Text（文本）编辑器中查看这些源代码，在 Project Explorer 中右击 Form1.java，在出现的快捷菜单中单击 View Code（查看代码）。具有 main 方法的程序开始执行。initForm 方法中的代码反应了在 Forms Designer 中所做的修改。

## 在 Forms Designer 中修改窗体

下列过程显示了如何在 Forms Designer 中将控件添加到窗体中。只要按钮被单击，该示例将显示一个列表框。

### 将控件添加到窗体

1. 在 Project Explorer 中的 Form1.java 上双击在 Forms Designer 中打开的窗体。

2. 在 Toolbox（工具箱）中选择 WFC Controls（WFC 控件）选项卡（如果工具箱没有显示，在 View（视图）菜单中单击 Toolbox。
3. 要将列表框添加到窗体，单击工具箱中的 ListBox（列表框）按钮，然后单击该窗体。

4. 要将按钮添加到窗体，单击在工具箱中的 Button（按钮）按钮，然后单击该窗体。

**设置控件属性**

1. 在窗体上选择 Button 控件，则在 Properties（属性）窗口中显示该按钮的属性[如果 Properties 窗口没有显示，则单击 View 菜单上的 Properties Window（属性窗口）]。
2. 找到按钮的 text（文本）属性，并且将该值改变为 Add Item（添加项目）。

**添加事件处理程序**

1. 也可以使用 Properties 窗口来将处理程序添加到控件的事件中。单击在 Properties 窗口中的 Events（事件）工具栏按钮来显示该 Button 控件的事件。
2. 找到 Click（单击）事件，并且输入 addItemClick 作为将处理该事件的方法的名字。当按下 Enter 键之后，Text 编辑器打开一个名为 addItemClick 的空的事件处理程序。
3. 在 addItemClick 事件处理程序的定义里，添加下面这一行代码：

```
listBox1.addItem("New string.");
```

有关在开发环境中修改代码的更多信息，见本书的第 3 章。

## 编译并运行应用程序

当使用 Windows Application 模板编译项目时，将自动创建一个名为 ProjectName.exe 的 .exe 文件。

### 编译并运行应用程序

1. 在 Build（编译）菜单上，单击 Build。在 Task List（任务列表）中会出现所有的编译错误或消息（在 Task List 中的某个错误上双击会将插入点（光标）移动到 Text 编辑器中该错误处）。改正该错误并且重新编译应用程序。
2. 要从开发环境中运行该应用程序，单击在 Debug（调试）菜单上的 Start（开始）。要在命令行中运行该应用程序，只需简单地运行这个 .exe 文件。
3. 单击 Add Item 按钮，一行文本出现在该列表框中。

4. 要关闭该应用程序，单击位于窗体右上角的 Windows Close（关闭）按钮。

有关使用 Forms Designer 和创建 WFC 应用程序的更多信息，请参见本书的第 2 章和第 11 章。

# 使用Application Wizard创建Windows应用程序

Application Wizard（应用程序向导）自动生成使用WFC的Windows应用程序，并且提供将窗体与数据库中的字段绑定的选项。还可以为项目指定封装选项。

> **注意：**在运行Application Wizard之前，关闭所有打开的项目（在File菜单中，单击Close All）。

**使用Application Wizard创建Windows应用程序**

1. 在File菜单上单击New Project（新项目）。
2. 在New选项卡上，展开Visual J++ Projects文件夹并单击Application（应用程序）。然后选择Application Wizard图标。
3. 在Name框中，输入项目名称。
4. 在Location框中，输入要保存该项目的路径，或单击Browse（浏览）按钮来定位到某个文件夹。
5. 单击Open（打开）。
6. 在Welcome（欢迎）步骤上，可以从下拉列表中选择某个配置文件[如果所需要的配置文件没有列出，单击省略号按钮（...）来找到并打开该配置文件]。有关配置文件的更多信息，参见本书的第5章。
7. 单击Next（下一步）来指定该应用程序的类型：
   - 选择Form based Application（基于窗体的应用程序）来生成Windows基于窗体的应用程序。

- 选择 Form based Application with Data（带有数据的基于窗体的应用程序）来生成包含数据绑定窗体的 Windows 应用程序。该选项启动 Data form Wizard（数据窗体向导），来引导用户完成指定要绑定的数据库和字段的过程。

8. 单击 Next 来指定想要添加到窗体中的特性：
   - Menu（菜单）：向导添加一个预定义的菜单栏。
   - Edit（编辑）：向导添加一个填充整个窗口的客户区域的 Edit 控件；该 Edit 控件用来创建一个简单的“NotePad(记事本)”应用程序。
   - Toolbar（工具栏）：向导添加一个带有预定义按钮的工具栏。
   - Status Bar（状态栏）：向导添加一个状态栏。

9. 单击 Next 来指定源代码注释的类型：
   - Javadoc Comment（Javadoc 注释）：向导将 Javadoc 注释添加到所有类和它们的成员中。这些注释提供有关生成代码的有用信息。
   - TODO Comment（TODO 注释）：向导在能够修改代码或增加的代码将增强应用程序的地方添加 TODO 注释。TODO 注释能在 Task List 窗口中查看。
   - Sample Functionality Comment（样本功能注释）：向导为其所插入的代码加上注释。这些注释提供有关插入代码的作用，并指出哪些代码是向导自动生成的。

10. 单击 Next 来指定封装选项：
    - Class file（类文件）：向导不将该项目放入任何类型的封装文件中。

如果想要使用除了封装以外的方法来发布应用程序，则选择该选项。

- EXE file（EXE 文件）：当编译项目时，创建一个.exe 文件。如果想要创建能够用来运行应用程序的单独的文件，选择该选项。
- Cabinet(CAB) file（CAB 文件）：当编译项目时，创建一个.cab 文件。.cab 文件是包含项目中所有重要信息的压缩文件。如果想要通过 Internet 发布应用程序，选择该选项。
- Deploy to (URL)（发布到）：可以定义一个 URL 来发布应用程序。在编辑框中指定 URL。

11. 单击 Next 查看应用程序的摘要：
    - 要查看自己的设置，单击 View Report（查看报告）。要保存该报告，单击 View Report 对话框中的 Save（保存）按钮。
    - 要保存这些设置到已存在的配置文件，从下拉列表中选择该配置文件。要保存这些设置到一个新的配置文件中，单击省略号按钮（...）来指定文件名（有关配置文件的更多的信息，参见本书的第 5 章）。
12. 单击 Finish（完成）按钮来创建该项目。应用程序在 Forms Designer 中被打开。

## 编译并运行应用程序

当编译了应用程序之后，就可以从程序开发环境中运行它。

### 编译并运行应用程序

1. 在 Build（编译）菜单上，单击 Build。在 Task List（任务列表）中会出现所有的编译错误或消息（在 Task List 中的某个错误上双击会将插入点（光标）移动到 Text 编辑器中该错误处）。改正该错误并且重新编译应用程序。
2. 要从开发环境中运行该应用程序，单击在 Debug（调试）菜单上的 Start（开始）。要在命令行中运行该应用程序，只需简单地运行这个 .exe 文件。
3. 单击位于窗体右上角的 Windows Close（关闭）按钮关闭应用程序，。

有关创建 Windows 应用程序的更多信息，还有一个显示如何在 Forms Designer 中修改窗体的简单的例子，见本章前面的“使用 WFC 创建 Windows 应用程序”一节。

## 创建动态 HTML 应用程序

WFC 隐藏了在 Internet Explorer 4.0 中实现的 Dynamic HTML（动态 HTML）对象模型。当使用 Code-Behind HTML 模板创建 DHTML 应用程序时，Java 类展开 com.ms.wfc.html.DhDocument，并且作为一个 COM 对象寄放在 HTML 页上。使用在 com.ms.wfc.html 软件包中的其他类，用户可以在 HTML 页上操作元素。

> **注意：** 在运行 Application Wizard 之前，关闭所有打开的项目（在 File 菜单中，单击 Close All）。

### 创建 DHTML 应用程序

1. 在 File 菜单上单击 New Project 命令。
2. 在 New 选项卡上，展开 Visual J++ Projects 文件夹，并单击 Web Pages。然后选择 Code-Behind HTML 图标。
3. 在 Name 框中输入项目的名称。
4. 在 Location 框中，输入要保存该项目的路径，或单击 Browse 按钮来定位到某个文件夹。
5. 单击 Open，一个项目的折叠视图出现在 Project Explorer 中。
6. 在 Project Explorer 中，展开该项目的节点，带有默认名字 Class1.java 的 Java 源文件已经添加到该项目中，还有一个名为 Page1.htm 的 HTML 页。

   **注意：** 将文件 Class1.java 改名并不会改变在源代码中与其关联的类名称，反之亦然。用户必须手工改变旧名字的所有实例[注意，可以创建一个空项目并且使用 Class（类）模板添加一个类。这两步过程允许用户在创建之前命名该类；无论如何，Class 模板并不为 DHTML 类提供基本代码框架]。

## 在 Text编辑器中查看代码

Code-Behind HTML 模板已经提供了一些样本功能。用户可以在 Text 编辑器中查看这些样本代码。

**在 DHTML 类中查看这些简单代码：**

- 要查看这些代码，在 Project Explorer 中双击 Class1.java。
  initForm 方法用于初始化代码，如在类中绑定元素到 HTML 页上现有的元素和创建新元素等。本示例实现将名为 boundText 的 DhText 对象绑定到 Page1.htm 页上现有的元素（<SPAN id=bindText>Bound Text</SPAN>）。这是通过首先调用 boundText.setId("bindText")，然后调用 setBoundElements 来完成的。这个示例还使用 setNewElements 方法添加新的 DhText 对象到该文档。

有关在开发环境中如何修改代码的信息，见本书的第 3 章。

## 编译并运行DHTML应用程序

因为 Class1 作为 COM 对象寄放在 HTML 页上，项目必须封装进一个.cab 文件中。通过使用 Code-Behind HTML 模板，在编译项目时，将自动创建名为 ProjectName 的.cab 文件。
一旦已经编译了项目，用户就可以从开发环境启动关联的 HTML 页，或只需在浏览器中打开 HTML 页。

**编译并运行 DHTML 应用程序**

1. 在 Build（编译）菜单上，单击 Build。在 Task List（任务列表）中会出现所有的编译错误或消息（在 Task List 中的某个错误上双击会将插入点（光标）移动到 Text 编辑器中该错误处）。改正该错误并且重新

编译应用程序。

2. 要从运行开发环境中运行该应用程序，在 Debug 菜单中单击 Start(开始)命令。Internet Explorer 启动并且 Page1.htm 从 DHTML 类中显示该元素。

有关使用 DHTML 的更多信息，见本书的第 14 章。

## 创建 COM DLL

当 Java 类封装到 COM DLL 中时，它可以被任何支持 COM 的应用程序所使用。在类中定义的所有公共方法通过 COM 接口展示出来。

> **注意：** 在运行 Application Wizard 之前，关闭所有打开的项目（在 File 菜单中，单击 Close All）。

**创建 COM DLL**

1. 在 File 菜单上单击 New Project 命令。

2. 在 New 选项卡上，展开 Visual J++ Projects 文件夹，并单击 Web Pages。然后选择 COM DLL 图标。
3. 在 Name 框中输入项目的名称。
4. 在 Location 框中，输入要保存该项目的路径，或单击 Browse 按钮来定位到某个文件夹。
5. 单击 Open，一个项目的折叠视图出现在 Project Explorer 中。

6. 在 Project Explorer 中，展开该项目的节点，带有默认的名字 Class1.java 文件已经添加到该项目中。

   **注意：** 将该文件改名并不会改变在源代码中与其关联的类名称，反之亦然。必须手工改变旧名字的所有实例[注意，可以创建一个空项目并且使用 Class（类）模板添加一个类。这两步过程允许用户在创建之前命名该类；无论如何，Class 模板并不为 COM DLL 类提供基本代码框架]。

## 在 Text编辑器中添加代码

在 Project Explorer 中双击 Class1.java 可以查看生成的源代码。@com,register 指令为类指定一个 GUID。还可以添加 onCOMRegister 方法到类中，该方法当 DLL 注册时自动调用；用户可以使用该方法来执行附加的注册，如创建任何 DLL 所需要的定制注册表键。
下列过程显示了如何添加构造器并创建一个显示消息框的方法。

### 添加代码到类中

- 在类声明中插入下列代码：

```
public class1()
{
}

public void showDialog()
```

```
{
    com.mswfc.ui.MessageBox.show("Hello, World!", "COM");
}
```

有关在开发环境中修改代码的更多信息，见本书第 3 章。

## 编译 DLL

当编译项目时，DLL 和类型库（分别名为 ProjectName.dll 和 ProjectName.tlb）将自动创建并注册到计算机上。

**创建 DLL**

- 在 Build 菜单上，单击 Build 命令。所有编译错误或信息出现在 Task list 中（在 Task list 中的错误上双击可以将插入点移动到 Text 编辑器中的错误上）。改正错误并重新编译。

## 导入 DLL

一旦 DLL 注册了，它就能够被任何支持 COM 的应用程序调用。下列过程显示如何导入 DLL 到其他 Visual J++项目中。该示例创建一个 WFC 窗体，在单击该窗体时调用 showDialog 方法。

**创建 WFC 应用程序**

1. 单击 File 菜单中的 Close All 命令来关闭 DLL 项目。
2. 要创建一个新项目，单击 File 菜单中的 New Project 命令。在 New 选项卡上，展开 Visual J++ Projects 文件夹，并单击 Applications。然后选择 Windows Application 图标。
3. 在 Name 框中输入项目的名称。在 Location 框中，输入要保存该项目的路径，或单击 Browse 按钮来定位到某个文件夹。
4. 单击 Open，在 Project Explorer 中展开该项目节点。一个带有默认的文件名 Form1.java 的文件已经添加到项目中。

**导入 DLL**

1. 在 Project Explorer 中，右击新项目的名称。在快捷菜单上指向 Add（添加），然后单击 Add COM Wrapper[添加 COM 封装(Wrapper)]。
2. COM Wrappers（COM 封装）对话框列出了注册到计算机上的类型库列表。选择在前面创建的 DLL 项目的名称。
3. 单击 OK。

DLL 作为一个子文件夹导入到新项目中，该文件夹包含两个.java 文件：Class1.java 和 class1_Dispatch.java。Class1 实现 Class1_Dispatch 接口，它列出 DLL 中 Class1 对象的公共方法。要访问这些方法，使用一个 Class1_Dispatch 对象来实例化 Class1。在下列过程中将演示这些内容。

**修改 WFC 窗体**

1. 在 Project Explorer，展开项目节点。在 Project Explorer 中双击 Form1.java 来在 Forms Designer 中打开它。

2. 在 Properties 窗口中单击 Events（事件）工具栏按钮来显示该窗体的事件（如果 Properties 窗口没有显示，在 View 菜单中单击 Properties Windows 命令）。
3. 找到 Click 事件并输入 formClick 作为将处理该事件的方法名。当按下回车键之后，Text 编辑器打开一个名为 formClick 的空的事件处理程序。
4. 转到该文件的开始部分并且添加下列 import 语句：

   ```
   import DLLProjectName.*;
   ```

   这里的 DLLProjectName 是所要导入的 DLL 项目名称。
5. 在 Form1 类定义中，在构造器的前面，声明一个 Class1_Dispatch 接口对象：

   ```
   Class1_Dispatch c;
   ```

6. 在 Form1 构造器中的 //T0D0 注释之后，通过 Class1_Dispatch 对象实例化 Class1。

   ```
   c = new Class1()
   ```

7. 在 formClick 方法定义中，通过 Class1_Dispatch 对象调用 Class1 的 showDialog 方法：

   ```
   c.showDialog()
   ```

**编译并运行 WFC 应用程序**

1. 在 Build 菜单上，单击 Build 命令（如果接收到任何编译错误或消息，改正错误并重新编译应用程序）。
2. 要运行该应用程序，在 Debug 菜单上单击 Start 命令。
3. 单击该窗体。一个消息框出现并显示“Hello, World!”。
4. 要关闭该应用程序，单击在窗体右上方的 Windows Close 按钮。

有关创建 COM 对象的更多信息，见本书的第 17 章。有关 WFC 应用程序的更多信息，见本章前面的“使用 WFC 创建 Windows 应用程序”一节。

## 创建控件

使用 WFC 可以创建自己的控件。使用 WFC 创建控件将展开 com.ms.wfc.ui. UserControl，它将依次展开 com.ms.wfc.ui.Form；这些可以使用 Forms Designer 来创建控件的用户界面。

> **注意**：在使用下列过程创建控件之前，关闭所有打开的项目（在 File 菜单中，单击 Close All）。

### 创建控件

1. 在 File 菜单上，单击 New Project 命令。
2. 在 New 选项卡上，展开 Visual J++ Project 文件夹，并单击 Components。然后选择 Control 图标。

3. 在 Name 框中输入项目的名称。
4. 在 Location 框中，输入要保存该项目的路径，或单击 Browse 按钮来定位到某个文件夹。
5. 单击 Open 按钮。一个折叠的项目的视图出现在 Project Explorer。
6. 在 Project Explorer，展开该项目的节点。一个带有默认名字 Control1.java 的文件已经添加到项目中。

**注意：**将该文件改名并不会改变在源代码中与其关联的类名称，反之亦然。必须手工改变旧名字的所有实例[注意，可以创建一个空项目并且使用 Class（类）模板添加一个类。这两步过程允许用户在创建之前命名该类]。

## 在 Text编辑器中查看代码

要查看所生成的源代码，需要在 Project Explorer 中右击 Control1.java，并且单击在快捷菜单上的 View Code（查看代码）命令。在 initForm 方法中的代码显示了在 Forms Designer 中所做的修改。嵌套的类 ClassInfo 指定控件的属性及事件[可以使用 WFC Component builder（WFC 组件编译器）来修改 ClassInfo]。

注意，用于 Control1 类的 Java 文档注释提供了如何作为 ActiveX 来公布类。更多信息见本章后面的“设置 COM 类”一节。

## 在Forms Designer中修改控件

下列过程显示了如何使用现有的 WFC 控件来设计控件。该例向控件添加了一个编辑框和一个水平滚动条；当滚动条滚动时，编辑框中显示滚动条当前的位置。

### 添加现有的控件到控件

1. 在 Project Explorer 中双击 Control1.java 来在 Forms Designer 中打开该控件。
2. 在工具箱中，WFC Controls 选项卡（如果工具箱没有显示，单击 View 菜单上的 Toolbox 命令）。
3. 单击在工具箱中的 Edit 控件并单击控件的设计表面来添加一个编辑框。
4. 单击在工具箱中的 HScrollBar 控件并单击控件的设计表面来添加一个滚动条。
5. 将编辑框的位置安排在控件设计表面的左上角，并且将滚动条移动到编辑框的下面（要移动某项，需要选中该项并且使用鼠标拖动它）。
6. 在编辑框和滚动条的周围重新调整控件的设计表面，以消除额外的空间（要设置设计表面的大小，需要选中设计表面并且拖动调整大小控制柄）。

### 添加事件处理程序

1. 单击 Properties 窗口中的 Event 工具栏按钮（如果 Properties 窗口没有

显示，单击 View 菜单上的 Properties Windows 命令）。

2. 要显示滚动条的事件，可以在控件的设计表面上选择滚动条，也可以在 Properties 窗口中的下拉式菜单中选择滚动条对象的名字。
3. 找到 scroll 事件并且输入 displayPosition 作为将处理该事件的方法名。当按下回车键之后，Text 编辑器打开一个名为 displayPosition 的空的事件处理程序。
4. 在 displayPosition 事件处理程序中，添加下面的一行代码：

```
edit1.setText ( "Position = " + HScrollBar1.getValue() );
```

有关在开发环境中修改代码的更多信息，见本书第 3 章“编辑代码”。

## 编译控件

当编译项目时，对于工具箱，控件自动成为可用状态。

**编译控件**

- 在 Build 菜单上单击 Build 命令。所有的编译错误或消息出现在 Task list 中（双击在 Task list 中的某个错误可以将插入点移动到 Text 编辑器中该错误的位置上）。改正该错误并重新编译控件。

**添加控件到工具箱中**

1. 右击工具箱并单击在快捷菜单上的 Customize Toolbox（定制工具箱）命令。

2.单击 WFC Controls 选项卡并选择控件名。
3.单击 OK。

## 添加控件到窗体中

要使用控件，可以添加它到 WFC 窗体中。

### 添加窗体到项目中

1.在 Project Explorer 中，右击所要添加的项目名称。在出现的快捷菜单上指向 Add，然后单击 Add Form（添加窗体）。
2.选择 Form 图标。
3.在 Name 框中，输入用于 Java 类的名称。
4.单击 Open。一个带有默认名字 Form1.java 的文件已经添加到该项目中，并且已经在 Forms Designer 中打开。

### 添加控件到窗体中

1.可以单击在 Toolbox 选项卡上的向上或向下箭头滚动控件列表，来在工具箱中找到该控件。
2.在工具箱中单击控件并单击该窗体来添加该控件。

### 创建并运行该窗体

1.在 Build 菜单中单击 Build 命令（如果收到一些编译错误或信息，改正

这些错误并重新编译项目）。

2.要运行该窗体，单击在 Debug 菜单上的 Start 命令。

**注意：**因为正在第一次运行项目，并且又因为该项目中包含两个.java 文件，则 Project Properties（项目属性）对话框显示出来。在 Launch（启动）选项卡上，选择 Default（默认）选项，并且指定当项目运行时将加载 Form1。单击 OK（有关项目属性的更多信息，见本章后面的“设置项目选项”一节）。

3.单击控件中滚动条的尾部，则在编辑框中显示当前的滚动位置。

4.单击位于窗体右上角的 Windows Close（关闭）按钮来关闭该窗体。

有关创建 WFC 控制和应用程序的详细信息，请参见第 13 章和第 11 章。

## 创建控制台应用程序

控制台应用程序是一个非图形用户界面。可以使用在 WFC 或 Java API 中的非图形类来开发应用程序。

**注意：**在使用下列过程创建控制台应用程序之前，首先应关闭所有已经打开的项目（在 File 菜单上选择 Close All 命令）。

**创建控制台应用程序**

1.在 File 菜单上，单击 New Project 命令。

2.在 New 选项卡上，展开 Visual J++ Projects 文件夹，并单击

Applications。然后选择 Console Application 图标。

3. 在 Name 框中输入项目的名称。
4. 在 Location 框中，输入要保存该项目的路径，或单击 Browse 按钮来定位到某个文件夹。
5. 单击 Open，一个项目的折叠视图出现在 Project Explorer 中。
6. 在 Project Explorer 中，展开该项目的节点，带有默认的名字 Class1.java 的文件已经添加到该项目中。

> **注意：** 将该文件改名并不会改变在源代码中与其关联的类名称，反之亦然。必须手工改变旧名字的所有实例（注意，可以创建一个空项目并且使用 ClassMain 模板添加一个类。这两步过程允许用户在创建之前命名该类）。

## 在 Text编辑器中添加代码

要查看所生成的代码，可以在 Project Explorer 中双击 Class1.java。程序执行使用 main 方法来开始。

### 添加代码到应用程序中

- 在 Text Editor 中，将下列代码添加到在 main 方法中的 //T0D0 注释之后。

```
String str = "The quick brown fox jumped over the lazy yellow dog. ";

System.out.println("The string is: " + str);
```

```
System.out.println("The length of the string is: "+ str.length());
System.out.println("The substring foom positions 10 to 20 is: "+ str.substring(10,20);

System.out.println("The uppercase string is: " + str.toUpperCase());
```

有关在开发环境中修改代码的更多信息，见本书的第 3 章“编辑代码”。

## 编译并运行该应用程序

当编译应用程序之后，可以从开发环境中或是从命令行中运行该应用程序。

### 编译并运行应用程序

1. 在 Build 菜单上，单击 Build 命令。所有的编译错误或信息出现在 Task list 中（在 Task list 中的错误上双击可以将插入点移动到 Text 编辑器中的错误上）。改正错误并重新编译应用程序。
2. 单击 Debug 菜单中的 Start 命令来在开发环境中运行应用程序。
3. 使用 JVIEW 命令从命令行中运行应用程序。在命令提示符上，从项目所在的目录位置上输入 jview Class1。

## 创建小应用程序

小应用程序从 HTML 页中运行，并且使用在 Java API 中的类来创建。小应用程序必须展开 java.applet.Appelt，并且通常使用在 java.awt 软件包中

的类来提供图形用户界面。

**注意：**在使用下列过程创建小应用程序之前，关闭所有已经打开的项目（在 File 菜单中，单击 Close All 命令）。

**创建小应用程序**

1. 在 File 菜单上，单击 New Projects 命令。
2. 在 New 选项卡上，展开 Visual J++ Project 文件夹，并单击 Web Pages。然后选择 Applet on HTML 图标。
3. 在 Name 框中输入项目的名称。
4. 在 Location 框中，输入要保存该项目的路径，或单击 Browse 按钮来定位到某个文件夹。
5. 单击 Open，一个项目的折叠视图出现在 Project Explorer 中。
6. 在 Project Explorer 中，展开该项目的节点，带有默认的名字 Applet1.java 的 Java 源文件已经添加到该项目中，同样也添加了一个名为 Page1.htm 的 HTML 页。

**注意：**将这个 Applet1.java 文件改名并不会改变在源代码中与其关联的类名称，反之亦然。必须手工改变旧名字的所有实例（注意，可以创建一个空项目并且使用 Class 模板添加一个类。这两步过程允许用户在创建之前命名该类；可是，Class 模板不会为小应用程序提供基本代码框架）。

## 在Text编辑器中查看代码

Applet模板已经提供了一些样本功能。用户可以在Text编辑器中查看这些样本代码。

### 查看小应用程序中的样本代码

1. 在Project Explorer中双击Applet.java来查看这些源代码。程序使用init方法来执行开始。
2. init方法仅仅调用initForm和usePageParams。使用Class Outline（类大纲视图）来定位initForm方法：
   - 如果Class Outline没有显示，在View菜单上指向Other Windows（其他窗口），并单击Document Outline（文档大纲视图）命令。
   - 在Class Outline中展开类，并且双击initForm。则插入点立即移动到initform方法上。
3. initform方法初始化背景和前景颜色，并且添加一个java.awt.Label控件到小应用程序中。
4. 在Class Outline中，双击usePageParams。该方法接收从关联的HTML页中的<PARA>选项卡的值，并且设置前景和背景颜色，还标注文本到这些值。如果usePageParams不能接收到<PARA>值，则使用默认值。

有关在开发环境中修改代码的更多信息，见本书第3章“编辑代码”。

## 编译并运行小应用程序

当编译小应用程序之后，可以从开发环境中启动相关的 HTML 页，或简单地在浏览器中打开该 HTML 页。

**编译并运行小应用程序**

1. 在 Build 菜单上，单击 Build 命令。所有的编译错误或信息出现在 Task list 中（在 Task list 中的错误上双击可以将插入点移动到 Text 编辑器中的错误上）。改正错误并重新编译应用程序。
2. 单击在 Debug 菜单中的 Start 命令来在开发环境中运行应用程序。

**注意：**要运行这个小的应用程序而不运行 HTML 页，可以在命令行中使用 JVIEW 命令。从该项目所在目录中的命令提示符上输入 jview /a Applet1。当指定/a 选项时，将启动 Applet View（小应用程序视图）来显示这个小应用程序。

## 创建空项目

用户可以通过使用某种代码模板、运行 Application Wizard 或创建空项目的方法来在 Visual J++中创建项目。当使用代码模板或向导来创建项目时，所生成的 Java 源文件使用默认的名字，如 Form1.java 或 Class1.java 等。将该文件改名并不会改变在代码中与其关联的类名称，反之亦然。用户必须手工改变旧名字的所有实例。

可是，创建空的项目则在添加 Java 源文件时提供了让用户命名该源文件的灵活性。

**注意：**在使用下列过程创建空项目之前，关闭所有已经打开的项目（在 File 菜单中，单击 Close All 命令）。

**创建空项目**

1. 在 File 菜单上，单击 New Project 命令。
2. 在 New 选项卡上，展开 Visual J++ Projects 文件夹。然后选择 Empty Project 图标。
3. 在 Name 框中输入项目的名称。
4. 在 Location 框中，输入要保存该项目的路径，或单击 Browse 按钮来定位到某个文件夹。
5. 单击 Open，该项目出现 Project Explorer 中，并且没有包含文件。

**添加窗体或控件到该项目中**

1. 在 Project Explorer 中，右击所创建的空项目名。在快捷菜单上指向 Add，然后单击 Add Form（添加窗体）命令。
2. 要添加一个 WFC 窗体，选择 Form 图标。Java 类展开 com.ms.wfc.ui.Form（还可以选择 Data Form Wizard 图标来自动生成一个数据绑定窗体）。
3. 要添加控件，选择 Control 图标。Java 类展开 com.ms.wfc.ui.UserControl。

4.在 Name 框中，为该 Java 类输入一个名称。
5.单击 Open。

有关 WFC 窗体的更多信息，还有显示如何修改窗体的简单例子，见本章前面的“使用 WFC 创建 Windows 应用程序”一节。

**添加 Java 类到该项目中**

1.在 Project Explorer 中，右击所创建的空项目名。在快捷菜单上指向 Add，然后单击 Add Class（添加类）命令。
2.要添加一个空的 Java 类，选择 Class 图标（可以一会再修改该类来创建 COM DLL、Dynamic HTML 应用程序或小应用程序）。
3.要添加一个包含 main 方法的 Java 类（就像创建一个控制台应用程序一样），选择 ClassMain 图标。
4.在 Name 框中，为该 Java 类输入一个名称。
5.单击 Open。

**添加 HTML 页到该项目中**

1.在 Project Explorer 中，右击所创建的空项目名。在快捷菜单上指向 Add，然后单击 Add Web Page（添加 Web 页）命令。
2.选择 Page 图标来添加一个 HTML 页。
3.在 Name 框中，为该页输入一个名称。
4.单击 Open。

### 导入 COM DLL 到该项目中

1. 在 Project Explorer 中，右击所创建的空项目名。在快捷菜单上指向 Add，然后单击 Add COM Wrapper（添加 COM Wrapper）命令。
2. 在 COM Wrappers 对话框中，选择要导入的库的类型。
3. 单击 OK。

> **注意：** 在默认情况下，当使用 Empty Project 图标来创建项目时，所创建的项目在编译时不会进行封装。可是，如果使用 WFC 创建 Windows 应用程序，用户可能希望将所创建的项目封装到一个 .exe 文件中，以便可以从命令行中运行它；如果创建一个 COM DLL，用户需要以 DLL 格式来封装该项目。更多信息见本书的第 7 章“封装和展开项目”。

## 创建多项目解决方案

每个所创建的项目都包含一个解决方案，但一个解决方案能够包含多个项目。

> **注意：** 在使用下列过程创建多项目解决方案之前，关闭所有已经打开的项目（在 File 菜单中，单击 Close All 命令）。

### 在解决方案中创建第一个项目

1. 在 File 菜单上，单击 New Projects 命令。
2. 在 New 选项卡上，展开 Visual J++ Project 文件夹，并单击 Applications，

然后选择 Windows Application 图标。

3. 在 Name 框中输入项目的名称。
4. 在 Location 框中，输入要保存该项目的路径，或单击 Browse 按钮来定位到某个文件夹。
5. 单击 Open（打开）。项目的折叠视图出现在 Project Explorer 中。

**注意：** 因为当创建 Windows 应用程序项目时没有项目（或解决方案）打开，所以新的项目将创建到一个新的解决方案中。

**添加其他项目到解决方案中**

1. 在第一个项目打开的情况下，在 File 菜单上单击 New Project 命令。
2. 在 New 选项卡上，展开 Visual J++ Projects 文件夹，并单击 Web pages，然后选择 Code-Behind HTML 图标。
3. 在 Name 框中输入项目的名称。
4. 在 Location 框中，输入要保存该项目的路径，或单击 Browse 按钮来定位到某个文件夹。
5. 因为当前已经有一个打开的解决方案，则现在有两个选项：
   - Close current solution（关闭当前的解决方案）：关闭当前的解决方案并在一个新的解决方案中创建该项目。
   - Add to current solution（添加到当前的解决方案）：保持当前的解决方案打开，并且将新的项目添加到其中。
6. 选择 Add to current solution 选项并且单击 Open 按钮。在 Project Explorer 中显示项目的折叠视图。

# 编译多项目解决方案

当创建一个多项目解决方案时，用户可以在该解决方案中编译单一的项目或是在解决方案中编译所有的项目。

**在解决方案中编译单一的项目**

1. 在 Project Explorer 中，选择所要编译的项目名称。
2. 在 Build 菜单上，单击 Build ProjectName（编译‘项目名’）命令。

**在解决方案中编译所有的项目**

- 在 Build 菜单上，单击 Build Sulution（编译解决方案）命令。

当编译整个解决方案时，每个项目按照添加到该解决方案的顺序来编译。可是，如果某个项目的编译必须取决于其他项目是否已经编译，用户可以明确地改变项目编译的顺序。

**在解决方案中改变项目的编译顺序**

1. 在 Project Explorer 中，右击该解决方案的名称，并且在快捷菜单上单击 Property Pages（属性页）。
2. 在 Property Pages 对话框中的 Build Order（编译顺序）选项卡显示了当前的编译顺序。要改变某个项目的顺序，应选择该项目名，并且单击其中的一个 Move 按钮。
3. 当改变项目的编译顺序的工作完成时，单击 OK。

## 设置启动项目

当编译多项目解决方案时，第一个添加到解决方案中的项目设置为startup project（启动项目），并且在Project Explorer中以粗体显示。启动项目就是单击在Debug菜单上的Start命令时运行的项目。

### 改变启动项目

1.在Project Explorer，右击想要将其设置为启动项目的项目。
2.在快捷菜单上，单击Set as Startup Project（设置为启动项目）命令。

## 使用Project Explorer管理项目

Project Explorer窗口显示打开项目所属的项。在Visual J++中，项目是基于目录的。在项目中的每个文件或文件夹都相当于在硬盘上的文件或文件夹。添加文件到项目也就添加该文件到硬盘上的项目目录结构中，反之亦然。注意，用户可以从项目中删除一个文件，但不从硬盘上将其删除。
当创建或打开项目时，Project Explorer默认自动打开。单击View菜单上的Project Explorer命令可以手工打开Project Explorer。

## 选择项目视图

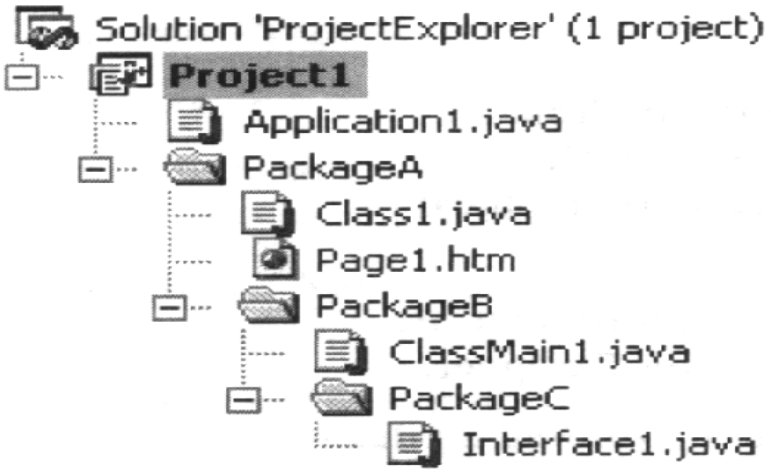
Java软件包实际上是在Java项目中的一个文件夹。在项目中Java软件

包的层次映射到在文件系统的文件夹的层次。因为这些关联，Project Explorer 提供两种方法来查看 Java 项目：目录视图和软件包视图。

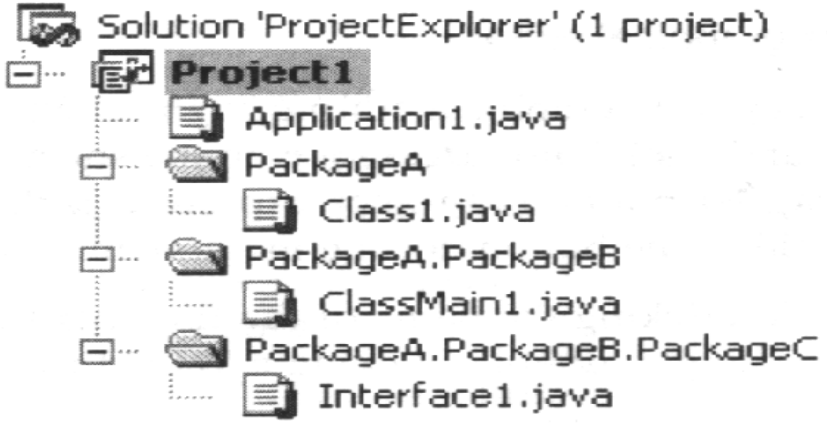
目录视图是默认视图，按照在硬盘上的项目目录结构显示所有项目子文件夹的分层目录结构。每个在该项目中的 Java 软件包显示为一个子文件夹。软件包视图在一个平面列表中将项目子文件夹显示为 Java 软件包格式。注意软件包视图只显示.java 文件和包含.java 文件的文件夹。每个文件夹按照全限定软件包名称列出。该项目节点本身作为默认的软件包。

下列图表显示了在目录视图和软件包视图中的相同项目。

**Directory View**




**Package View**




有关在 Java 软件包和文件系统之间关系的更多信息，见“Java language Specification”（Java 语言规范）中的“Storing Packages in a File System（在文件系统中存储软件包）”。

### 在软件包视图中查看项目

- 在 Project 菜单中，单击 Package View（软件包视图）命令或单击在 Project Explorer 中的 Package View 按钮。只有项目的.java 文件和包含.java 文件的文件夹（软件包）显示出来（如果某个非 Java 源文件正在打开，那么该文件也会在 Project Explorer 中显示）。

  **注意：**在软件包视图中显示的文件和文件夹的结构基于.java 文件的位置，而不是基于编译.java 文件的位置。因此，如果在.java 文件中的软件包说明与它们在硬盘上的位置不匹配，则软件包视图将不会正确显示项目中的软件包。

### 在目录视图中查看项目

- 在 Project 菜单上，单击 Directory View（目录视图）命令，或单击在 Project Explorer 中的 Directory View 按钮。在项目中的所有文件及文件夹将显示出来。

Project Explorer 默认不显示在硬盘上的项目目录结构中已有的所有非项目文件和文件夹。可是，如果项目视图设置为目录视图，则可以通过单击在 Project Explorer 中的 Show All files（显示所有文件）按钮来显示这些项。注意，在软件包视图中该按钮被禁用。有关在项目目录结构中存在的非项目项的更多信息，见本章后面的“在项目目录结构中显示所有文件”一节。

## 打开文件

可以使用 Project Explorer 来打开在 Java 项目中的文件。每个文件与一个源编辑器相关联。如果该文件支持图形编辑，那么它还与一个图形编辑器相关联。

### 在文件的源编辑器中打开文件

- 在 Project Explorer 中，右击该文件的名字，并且在快捷菜单上单击 View Code（查看代码）命令，或者单击在 Project Explorer 中的 View Code 按钮。

### 在文件的图形编辑器中打开文件

- 如果该文件支持图形编辑，在 Project Explorer 中右击该文件的名字，并在快捷菜单上单击 View Designer（查看设计）命令。或者单击在 Project Explorer 中的 View Designer 按钮（如果该文件不支持图形编辑，则该按钮及菜单都不可用）。

  **提示：** 双击将在该类型文件的默认编辑器中打开的文件的文件名。例如，双击一个 WFC 窗体将会在 Forms Designer 中打开该窗体。

有关添加文件到项目中的更多信息，见下一小节“添加文件”。

## 杂项文件

要打开不属于项目一部分的文件，使用在 File 菜单上的 Open File（打开

文件）命令。Project Explorer 在 Miscellaneous File 节点下列出这些文件。一旦一个非项目文件在 Miscellaneous File 下列出，可以在 Project Explorer 中双击该文件来重新打开它。

注意，如果从项目中删除一个文件，该文件将一直存在于项目目录结构中。可是，当该文件不再属于某个项目之后，用户必须一直使用 Open File 命令来打开该文件。有关删除文件的信息，见本章后面的“删除文件或文件夹”一节。

## 添加文件

当添加文件到项目中时，它被包含在该项目的编译过程中。添加文件到 Java 项目中也添加该文件到在硬盘上的项目目录结构中。

- 在项目中创建一个新文件也在磁盘上创建一个新文件。
- 添加一个磁盘上已有的文件到项目中，将复制该文件到项目目录结构中适当的位置。

  **注意：**某些文件可能存在于项目目录结构中却不属于该项目。例如，从项目中删除某个文件会将该文件留在项目目录结构中。该文件能够添加回到在磁盘的同样位置中。

可以将文件添加到项目的最高一级中或添加到指定的文件夹中。有关添加文件夹到项目的信息，见本章后面的“添加文件夹”一节。

**添加文件到项目中**

1. 在 Project Explorer 中，选择要添加文件的项目或文件夹节点。
2. 在 Project（项目）菜单上，单击 Add Item（添加项）命令。
3. 要创建新文件，在 Add Item 对话框中单击 New 选项卡。
   - 在对话框的左窗格中，选择文件种类。在右边的窗格中，基于左边窗格中的选择来选择所要添加的文件类型。
   - 在 Name 框中，为该文件输入文件名。有关有效的 Java 文件名的信息，见在“Java Language Specification”中的“Identifiers”一部分。
   - 单击 Open 按钮。注意当添加新的.java 文件时，如果该项目节点中包含默认的软件包，则 Visual J++自动在该文件中添加适当的软件包说明。
4. 要添加一个现有的文件，在 Add Item 对话框中单击 Existing（现有的）选项卡。
   - 找到并选择所需要添加的文件。可以使用 SHIFT 或 CTRL 键在一个文件夹中选择多个文件。
   - 单击 Open 按钮。注意，当添加一个现有的.java 文件时，用户必须手工改变它的软件包状态来关联到新的的软件包（注意，该项目节点本身已被视为默认软件包）。如果文件的软件包说明没有映射到它所存在的文件夹，当编译该项目时，输出的目录结构将不与源目录结构相匹配。

当一个文件通过 Project Explorer 添加到项目中时，它自动在它默认的浏览器中打开。当同时添加多个文件时，则没有文件打开。有关打开文件的信息，见本章后面的“打开文件”一节。

**注意：**如果文件通过文件系统添加到项目目录结构中时，如果该文件类型是在项目过滤器中指定的，那么它将自动添加到项目中。然后该文件在 Project Explorer 中显示，但没有打开。有关项目过滤器的更多信息，见本章后面的“在项目目录结构中显示所有文件”一节。

## 添加驻留于项目目录结构中的文件

如果文件不属于某个项目，但是它却存在于该项目的目录结构中，用户可以将其以同样的位置添加到项目中。

**添加驻留于项目目录结构中的文件**

1. 在 Project Explorer 中单击 Directory View 按钮将项目视图设置为目录视图。只有目录视图才允许用户看到存在于项目目录结构中的非项目文件。有关视图的更多信息，见本章前面的“选择项目视图”一节。
2. 如果该非项目文件没有正确显示，单击在 Project Explorer 中的 Show All Files 按钮来显示在项目目录结构中所有的文件。
3. 在想要添加到该项目的文件名字上右击。注意，如果文件所属的文件夹不在项目中，则不能添加这个文件。有关添加文件夹的信息，见本章后面的“添加驻留于项目目录结构中的文件夹”一节。
4. 在快捷菜单上，单击 Add To Project（添加到项目）命令。该文件添加到项目中，但没有打开。有关打开文件的信息，见前面一小节“打开文件”。
5. 要隐藏剩下的非项目文件，再次单击 Show All Files 按钮。

有关存在于项目目录结构中的非项目文件的信息，见本章后面的“在项目目录结构中显示所有文件”一节。

## 添加文件夹

当添加新的文件夹到项目中时，将在硬盘上创建一个相应的文件夹。在 Java 项目中，一个文件夹实际上是一个 Java 软件包。在 Project Explorer 的目录视图中，所有软件包做为项目子文件夹显示。在软件包视图中，子文件夹作为 Java 软件包显示。有关视图的更多信息，见本章前面的“选择项目视图”一节。

> **注意：** 文件夹可能存在于某个项目目录结构中，但不属于该项目。例如，从项目中删除某个文件夹会将该文件夹一直留在项目目录结构中。该文件夹可以以其原有的位置加回到该项目中。

### 添加文件夹（软件包）到项目中

1. 在 Project Explorer，右击将包含新文件夹的软件包或文件夹的节点。
2. 在快捷菜单中，单击 New Folder（新文件夹）。
3. 在 Add Package /Folder（添加软件包/文件夹）对话框中，输入新的文件夹名。该名称必须是一个有效的 Java 软件包名称。关于有效名称的信息，见“Java Language Specification”一书的“Identifiers”部分。
4. 单击 OK。

一旦创建文件夹，就可以将文件添加到其中。有关添加文件的信息，见

本章前面的“添加文件”一节。

> **注意：**如果文件夹通过文件系统添加到项目目录结构中，它自动添加到项目中，并在 Project Explorer 中显示出来。在该文件夹中，所有在项目过滤器指定文件类型的文件也添加到该项目中。有关项目过滤器的更多信息，见本章后面的“在项目目录结构中显示所有文件”一节。

## 添加驻留于项目目录结构中的文件

如果文件夹不属于某个项目，但是它却存在于该项目的目录结构中，用户可以将其以同样的位置中添加到项目中。

### 添加驻留于项目目录结构中的文件夹

1. 在 Project Explorer 中单击 Directory View 按钮将项目视图设置为目录视图。只有目录视图才允许用户看到驻留于项目目录结构中的非项目项。有关视图的更多信息，见本章前面的“选择项目视图”一节。
2. 如果该非项目项没有正确显示，单击在 Project Explorer 中的 Show All Files 按钮，来显示在项目目录结构中所有的文件和文件夹。
3. 在想要添加到该项目的文件夹名字上右击。
4. 在快捷菜单上，单击 Add To Project 命令。该文件夹及包含的所有子文件夹和文件添加到项目中（只有在项目过滤器中指定文件类型的文件才添加进来）。
5. 要隐藏剩下的非项目项，再次单击 Show All Files 按钮。

有关项目过滤器和有关驻留于项目目录结构中的非项目项的信息，见本章后面的“在项目目录结构中显示所有文件”部分。

## 移动或复制文件或文件夹

在 Java 项目中当移动或复制文件或文件夹时，在硬盘上的文件或文件夹也移动或复制。移动或复制文件夹也将移动和复制在该文件夹中的所有文件。

> **重点：**当移动或复制一个.java 文件到不同的文件夹时，用户必须手工改变它的软件包说明来引用这个新的软件包（注意，该项目节点本身已被视为默认软件包）。如果文件的软件包说明没有映射到它所存在的文件夹中，当编译该项目时，输出的目录结构将不与源目录结构相匹配。

当移动或复制项时，Project Explorer 遵从文件系统的规定。例如，不能将文件夹移动到它的子文件夹中。

### 在项目中移动或复制文件或文件夹

1. 在 Project Explorer 中，右击所要移动或复制的文件或文件夹名称。
2. 要移动该项，在快捷菜单上单击 Cut（剪切）命令；要复制该项，单击 Copy（复制）命令。
3. 右击新的文件夹或将包含这些文件或文件夹的项目节点。
4. 在快捷菜单上，单击 Paste（粘贴）命令。

**注意：**如果文件或文件夹是通过文件系统进行移动或被复制，它们将在 Project Explorer 中自动移动或复制。可是，用户必须确定在 .java 文件中的软件包说明必须是正确的。

还可以使用在 Project Explorer 中拖动要移动或复制的文件或文件夹的方法来移动或复制。注意，在 Project Explorer 中的拖放操作仅复制源项。在目录视图中，用户可以拖动单独的文件夹或拖动任意的文件集。在软件包视图中，用户只能拖动文件。关于视图的更多信息，见本章前面的“选择软件包视图”一节。

## 重命名文件或文件夹

当重命名在 Java 项目中的文件或文件夹时，在硬盘上的文件或文件夹也将重命名。

**重点：**当在项目中重命名项时，用户必须仔细地手工改变所有到原名称的引用。例如，Java 需要一个具有同样名字的公共类来作为它的源文件。那么，如果将一个 .java 文件重命名，那么必须重命名与其关联的公共类，反之亦然。

注意，如果重命名一个文件夹名称，必须改变在该文件夹中的所有 .java 文件的软件包说明。如果文件的软件包说明没有映射到它所存在的文件夹，当编译该项目时，输出的目录结构将不与源目录结构相匹配。

### 在项目中重命名文件或文件夹

1. 在 Project Explorer 中，右击将被重命名的文件或文件夹。
2. 在快捷菜单上，单击 Rename（重命名）命令。
3. 输入新的名称并且按下回车键。所输入的名字必须是一个有效的文件或软件包名称。有关有效名称的信息，见“Java Language Specification”一书的“Identifier”部分。

   **注意：**如果文件或文件夹通过文件系统重命名，则其自动在 Project Explorer 中重命名。可是，仍须改变所有到原名称的引用。

## 删除文件或文件夹

当从 Java 项目中删除文件时，该文件不再与项目一同编译；但是，该文件将留在硬盘上的该项目目录结构中。当删除文件夹时，该文件夹和文件夹中的所有文件从项目中删除，但它们还留在硬盘上。
可以选择从项目和硬盘上真正删除文件或文件夹。

**注意：**移走或删除一个 .java 文件并不自动将与其关联的 .class（如果该文件存在的话）文件移走或删除。但是，可以在 Project Explorer 中显示该 .class 文件，然后手工将其移走或删除。

### 从项目中移走文件或文件夹

1. 在 Project Explorer 中，右击所要移走的文件或文件夹名称。
2. 在快捷菜单上，单击 Remove from Project（从项目中移走）。

### 要从项目中删除文件或文件夹

1. 在 Project Explorer 中，右击所要删除的文件或文件夹名称。
2. 按下 DELETE 键。

**注意：** 如果某项已经通过文件系统删除，那么它将自动从项目中删除，并且从 Project Explorer 中移走。

Project Explorer 提供显示已从项目中移走的项的选项。更多信息见下一节“在项目目录结构中显示所有文件”。有关将移走的文件或文件夹添加回到项目中的信息，见本章前面的“添加文件”及“添加文件夹”两部分。

## 在项目目录结构中显示所有文件

文件或文件夹可以驻留于某个项目目录结构中，但它们不属于该项目：

- 从 Java 项目中删除文件或文件夹后仍将其遗留在硬盘上。
- 文件的类型没有在项目过滤器中指定，不能添加到项目中。更多信息见下一节“项目过滤器”。

在 Project Explorer 的目录视图中，有一个选项可以显示所有驻留于项目目录结构中的文件或文件夹，尽管它们有些不属于该项目。每个项的图标指出它们是否属于该项目。

### 在项目目录结构中切换到非项目项的显示

1. 单击在 Project Explorer 中的 Directory View 将项目视图设置为目录视

图。只有目录视图才允许查看驻留于项目目录结构中的非项目项。有关视图的更多信息，见本章前面的“选择项目视图”一节。

2.在 Project Explorer 中单击 Show All Files 按钮。

有关将非项目文件或文件夹添加到项目中的信息，见本章前面的“添加文件”及“添加文件夹”两部分。

## 项目过滤器

Visual J++使用项目过滤器来确定哪种文件类型属于 Java 项目。项目过滤器是一个包含过滤器，即过滤器使用扩展名来标识有效的项目文件。下面列出的扩展名包含在项目过滤器中：

- .java
- .asp , .htm , .html
- .bmp , .gif . .ico , .jpg , .jpeg
- .avi , .au, .wav
- .doc , .rtf, .txt , .ppt , .wri , .xls
- .bat , .cur , .reg , .snd , .tlb , .url

当文件从外部 Visual J++环境中添加到项目目录结构中时，使用项目过滤器来确定该文件实际上是否属于该项目。

> **重点：**当在 Visual J++环境中添加文件时，文件总是添加到该项目中，而不管该文件的类型是什么。

## 设置项目选项

Visual J++提供了许多选项，如允许用户为调试来配置应用程序、优化项目的编译输出、在项目中将类声明为 COM 对象以及指定类所从属的路径等。可以在〈Project〉对话框（其中的〈Project〉是当前所选定的项目）中找到所有的这些选项。可以通过单击 Project 菜单，并选择〈Project〉Properties 菜单命令来显示这些选项。

## 设置启动选项

在 Visual J++中编译项目之前，需要定义将要首先加载并作为应用程序的入口点的项目文件。选择该文件有两种方法，一是使用默认的程序和命令行从可用的项目文件列表中选择它；或者选择定制的程序和命令行。对于多数应用程序来说，默认的程序和命令行参数已经足够了。当需要定义一个定制的命令行传递到正在 Visual J++中运行的应用程序时，经常用到指定定制的程序和命令行。当正在开发需要从单独的程序中测试的 WFC 控件或 COM DLL 时，用户可以使用这些定制程序选项。

### 选择编译配置

1. 在 Project 菜单上，单击〈Project〉Properties（项目属性，其中的〈Project〉是当前所选定的项目）命令。

〈Project〉Properties 对话框出现。

2. 选择 Launch（开始）选项卡。

3. 在 Configuration 下拉列表中，为 Launch 选项卡设置要应用的部分选择编译配置。

**使用默认程序和命令行来选择入口点文件**

1. 在〈Project〉Properties 对话框中，单击 default 按钮来使用默认的选项。
2. 在 When project runs，load（当项目运行时，加载）下拉框中，选择想要用作应用程序的入口点的文件。下面是能够在该列表中显示的文件类型：
   - 定义有 main 方法的任何 Java 类。
   - HTML 文件。
   - 扩展 java.lang.Applet 的 Java 类。

   Program 文本框根据从入口点列表中选择的文件类型识别将开始应用程序的程序。Arguments（参数）文本框显示了当程序运行时传递的命令行参数。
3. 如果从入口点列表中选择的文件是一个 Java 源文件，可以选择 Launch as a console application（作为控制台应用程序启动）复选框来使用 JView 替代 WJView 运行应用程序。JView 在一个控制台窗口中显示所有应用程序的输出，而 WJView 在一个 Output（输出）窗口中显示所有的输出。

**选择入口点文件并定义定制程序和命令行**

1. 在〈Project〉Properties 对话框上，单击 Custom（定制）按钮。Program 和 Argument 文本框中包含在 Default 节中定义的同样的文件、程序和

参数。

2. 在 Program 文本框中，输入一个新的程序名来运行或编辑已有的程序名。
3. 在 Arguments 文本框中，输入新的参数列表来使用或编辑已有的参数列表。所提供的默认参数还包含将作为入口点的项目文件名的列表。

**注意：** 如果想要为项目的调试版本和发行版本配置不同的入口点文件，可以使用 Configuration 下拉列表来指定新的设置。

## 设置编译器选项

Visual J++提供了大量的编译器选项来帮助优化编译输出及在应用程序的调试中提供帮助。〈Project〉 Properties 对话框中的 Compile（编译）选项卡提供了设置这些选项的方法。在该对话框中所提供的多数选项是复选框，可以将某个特性设置为可用或禁用。对于在该对话框中每个选项的信息，可以单击位于该对话框中的 Help（帮助）按钮来查看。

在编译代码时，如果想要指定使用条件编辑符号，可以使用在 Compile 选项卡上的选项来定义它们。将条件编译符号输入到 Conditional compilation symblos（条件编译符号）框中，并使用逗号将每个符号分开。有关设置符号和条件编译的更多信息，见本书的附录 B“条件编译”。

除了可以在该对话框中改变的编译器选项，用户还可以定义当其他选项传递之后，传递到编译器的附加选项。还可以在 Additional compiler options（附加编译器选项）文本框中输入附加编译选项来指定它们。每

个选项将由空格分开。有关可用的编译器选项，见本书第 9 章中的“JVC 命令行选项”一节的“用于 JVC 编译应用程序”。

可以在 Compile 中指定的其他的编译设置是 Output Directory（输出目录）。该框允许用户指定一个目录，在该目录中放置项目的编译文件。如果项目中有一些在其中定义的软件包，Visual J++在输出目录中创建这些软件包，并且在这些软件包中编译所有的源文件。可以使用该设置来指定项目的调试或发行版本的编译位置。因为该框默认是空的，因此，所有项目输出默认放置到该项目的目录中。

> **注意：**Configuration 列表允许用户为项目的发行版本和调试版本具有不同的设置。可以使用该特性来指定特殊的编译器设置，并且为项目的每个配置指定一个输出目录。All Configurations 设置也可用于改变全部项目的配置。

## 创建定制编译准则

根据项目类型的不同，在编译工作的前后，可能需要完成一些任务。在〈Project〉属性对话框的 Custom Build 选项卡中，可指定在编译项目之前或之后执行的命令。在该选项卡上有两个文本框—Pre-build（预编译）和 Post-build（后编译）。输入到该文本框中的命令格式与用在批处理文件（如 Autoexec.bat）中的命令一样。

### 添加预编译或后编译命令

1.在 Project 菜单上，单击〈Project〉Properties（其中的〈Project〉是当

前在 Project Explorer 中所选定的项目）。
〈Project〉 Properties 对话框出现。
2. 选择 Custom Build 选项卡。
3. 在 Configuration 下拉列表中，为 Custom Build 选项卡设置所应用的部分选择编译配置。
4. 如果想要指定在项目编译之前发生的命令，选择 Pre-build command(s)（预编译命令）框。如果想要指定在项目编译之后发生的命令，选择 Post-build command(s)（后编译命令）框。
5. 输入想要为选定的命令类型指定的命令。在该文本框中的每行只能指定一个命令。当输入命令到文本框中时，可以使用标准的批命令指定执行该命令的条件。

## 设置类路径

要利用在项目中的 Java 软件包，重要的是该软件包在 Classpath 中是可用的。Classpath 是编译器用来查找项目中所涉及到的软件包的路径。可以通过下列两种方法之一将软件包添加到 Classpath 中：如果正在运行 Microsoft Windows，则将软件包的路径添加到 Autoexec.bat 中；如果正在运行 Windows NT，则将软件包的路径添加到环境设置中。下面列出了一个 Classpath 项的例子：

```
SET CLASSPATH = %PATH%,C:\MYPACKAGE
```

用户可能还想为 Java 项目添加路径到 Classpath。在〈Project〉 Properties

对话框中的 Classpath 选项卡允许添加或删除涉及项目的路径。

**显示 Classpath 选项卡**

1. 在 Project 菜单上，单击〈Project〉 Properties 命令（其中的〈Project〉是当前在 Project Explorer 中所选定的项目）。
〈Project〉 Properties 对话框出现。
2. 选择 Classpath 选项卡。
3. 在 Configuration 下拉列表中，为 Classpath 选项卡设置要应用的部分选择编译配置。

**添加路径到项目特定的 Classpath 中**

1. 在〈Project〉 Properties 对话框中，单击 New 按钮。
Extended project-specific Classpath（扩展项目指定类路径）对话框出现。
2. 输入希望添加到项目的 Classpath 中的路径。如果所定义的路径不存在，Visual J++将询问是否添加该路径而不管该路径是否存在。

3. 单击 OK。
该路径添加到路径列表中。

**注意：**如果在 Visual J++中有多个项目打开，可以将所有的项目特定的 Classpath 合并为一个统一的“项目特定的”Classpaths。方法是单击 Merge all Project-specific Classpaths in Solution（在解决方案中合并所有项目特定的 Classpath）复选框。

要从该列表中删除一个 Classpath，选择在该列表中的 Classpath，然后单击 Delete（删除）按钮。还可以使用 Up 或 Down 按钮来改变 Classpaths 的顺序。

## 设置 COM 类

公布某个类作为 COM 类是创建功能强大和可重用应用程序的方法。通过选择一个作为 COM 的类，该类及它的公共方法便能够被其他应用程序和编程语言所访问。在〈Project〉Properties 对话框中的 COM Classes 选项卡中，可以指定在项目中将作为 COM 类公布的公共类。一旦某个类已经定义为 COM 类，Visual J++为项目的 COM 类创建一个类型库，这样，该类就可以被其他应用程序访问，并且可以在系统的注册表中作为 COM 类来注册。

COM Classes 选项卡还允许用户使用一个已有的类型库来公布项目的类，或为已有的类型库或内含的 COM 对象 ActiveX 控件创建类模板。COM 模板允许使用 Java 源代码来执行在类型库或以前存在的 COM 组件中描述的 COM 组件。如果用户有想要使用 Java 执行的类型库或 COM 组件，则可以在列表中选择该类型库或 COM 组件，并且 Visual J++将在项目中编译模板代码。然后，可以执行由模板提供的每一个方法，并且完全创建一个 COM 组件。

**将项目中的类作为一个 COM 类**

1.在 Project 菜单上，单击〈Project〉 Properties（其中的〈Project〉是当

前在 Project Explorer 中所选定的项目）。
〈Project〉 Properties 对话框出现。

2. 选择 COM Classes 选项卡。

   **注意：**在〈Project〉Properties 对话框的 COM Classes 选项卡中的所有设置不受在 Configuration 列表中的编译配置设置的改变的影响。

3. 在 COM Classes 选项卡中，单击 Automatically generate Type Library（自动生成类型库）。
4. 在要作为 COM 类的类旁边放置一个选择标记。
5. 默认情况下，Visual J++根据项目的名称来创建一个类型库文件。要改变该名称和其他关联到该类型库的信息，如库名称、描述、帮助文件和与上下文有关的帮助 ID 等，可以单击 Options 按钮，此时显示出 Type Library Options（类型库选项）对话框，用户可以在其中修改该类型库的信息。

**使用已有的类型库**

1. 在 COM Classes 选项卡上，单击 Use existing Type Library（使用已有的类型库），然后单击 Select（选择）。Com Templates（COM 模板）对话框出现。
2. 从 Installed COM Components（已安装的 COM 组件）列表中，定位到所要使用的类型库上，并在其旁边放置一个选择标记。如果该类型库在系统注册表中没有注册，可以单击 Browse（浏览）按钮来选择所要

使用的类型库。

3.单击 OK。

Visual J++在项目目录中创建一个软件包目录，并且还在新的软件包目录中创建模板类。

## 导入Visual J++ 1.1版项目

用户可以在 Visual J++ 6.0 开发环境中打开并使用在 Visual J++ 1.1 中所创建的项目。下面的过程提供了转换这些文件的步骤。

> **小心：** 在导入 Visual J++ 1.1 的项目之前，重要的是应该了解这个新项目的结构。新项目基于目录的结构是与 Visual J++ 1.1 版本中基于文件的结构相对应的。在项目结构中的这种改变，意味着项目中根目录中已有的所有文件和在项目根目录下的所有目录都自动包含为项目的一部分。用户可以使用 Project Explorer 快捷菜单中的 Remove From Project（从项目中移出）项中从新项目中移出文件。如果已从项目中移出了某项，可以使用在 Project Explorer 快捷菜单中的 Add To Project（添加到项目中）项来将其加回到项目中。可是，当用户从项目中删除了某个文件时，该文件从目录中被实际删除，并且不能从开发环境中将其恢复，而只能从 Recycle Bin（回收站）中将该文件恢复。

**导入 Visual J++ 1.1 版项目**

1. 在 File 菜单上，单击 Open Project（打开项目）命令。
   出现 New Project 对话框。
2. 单击 Existing（已有）选项卡，并且找到所要导入的 Visual J++ 1.1 项目的根目录。
3. 单击 Visual J++ 1.1 项目的根目录。
4. 在 New Project 对话框中，单击 Import Project Folder（导入项目文件夹）按钮。
   所导入的 Visual J++ 1.1 项目出现在 Project Explorer 中。
5. 在 Project Explorer 中，展开项目的文件夹来查看在项目中的文件。

该项目现在已经准备好进行修改、编译和运行。

# 第 2 章　设计窗体

Forms Designer 提供了为应用程序可视创建窗体的动态方法。用户可以从二维窗体转换到基于 Windows Foundation Classes for Java（WFC，用于 Java 的 Windows 基础类）的源代码。Forms Designer 包含 Rapid Application Development（RAD，快速应用程序开发）特性，如代码生成和代码补全，并提供快速语法分析、调试信息和其他工具等，例如：

- 在无须写出 Java 源代码的情况下创建窗体，并指定对齐方式及格式。
- 管理包含 WFC、ActiveX 和定制控件的工具箱。
- 通过将 WFC 和 ActiveX 控件拖动到窗体上来将控件添加到窗体中。Forms Designer 生成适当的 Java 源代码。
- 将事件处理程序添加到控件中。
- 在无须写出 Java 源代码的情况下，通过拖动窗体和控件的控制柄和对象本身来改变它们的位置和大小。
- 在无须写出 Java 源代码的情况下，使用 Properties（属性）窗口来设置属性。Properties 包含了组成窗体的各个要素，如颜色、大小、位置和字体等。
- 通过附加单独的 WFC 控件为窗体创建菜单。
- 对代码改变和窗体视图的可视性，具有代码生成同步。

- 为窗体编写或编辑 Java 代码，并在 Forms Designer 中查看结果。
- 为窗体创作 WFC 控件。

## 创建窗体

用户可以通过 Project（项目）菜单将窗体添加到应用程序中。
包含组件和控件的对象可以从工具栏拖动到窗体中。该窗体一直作为定义 WFC Form 类实例的源代码块。默认的 Form 类包括：

- 激活 initForm 方法的构造器。
- 在窗体上的每一项的实例变量集。
- 为这些项指定默认属性的一个 initForm 方法。可以通过设置项属性来改变这些默认值。

Forms Designer 根据在每个控件和窗体本身的 Properties（属性）窗口中的设置来生成代码。当修改这个 initForm 方法时要注意，因为 Forms Designer 在显示该窗体时也要读取此代码。而且，要避免如 if，while 和 switch 这样的条件语句和组件方法。如果要修改 initForm 代码，可限定改变到以 set 起始的属性，而不限定值为常量。不要使用变量来设置值。
例如：

```
正确    :          setSize(10,100)
错误:          x=10 ; y=100 ; setSize(x,y)
```

**创建窗体**

1.打开一个已有的项目或创建一个新项目。
2.在 Project Explorer 中，选择项目的名称。
3.在 Project 菜单上，单击 Add Form（添加窗体）命令。
4.在 Add Item（添加项）对话框中，单击 New（新建）或 Existing（已有的）选项卡中的一个。在本例中，单击 New 选项卡。
5.在 Add Item 对话框中，单击 Form 图标。
该模板创建一个 capable of hosting controls 类。
6.在 Name 框中修改类文件的名称。默认的文件名为 Form1.java。
7.单击 Open 按钮。
该窗体窗口显示，并显示窗体表面。

如果正在从模板中创建一个新的窗体，该窗体自动以设计模式打开。如果 Forms Designer 当前未被激活，使用下列过程来打开它。

**在 Forms Designer 中查看窗体**

- 在 View 菜单上，单击 Designer 命令。
或者
双击在 Project Explorer 中的节点，或右击窗体文件名，然后在出现的快捷菜单中单击 View Designer（查看设计器）命令，窗体现在出现在 Forms Designer 中。

控件默认对齐到窗体网格线上。可以使用在 Options（选项）菜单中的 Tools（工具）命令来设置网格选项。窗体属性能够在该窗体的属性页上直接修改。

工具箱包含 WFC Controls（WFC 控件），并且包含其他包含 ActiveX 和定制控件的选项卡。
一旦有了基本的窗体模板，用户就可以添加控件到其中。

## 管理工具箱

Forms Designer 包含默认控件的工具箱。该默认控件的类型按选项卡分开。用户可以将定制的选项卡添加到工具箱中，并添加定制的控件到任何一个选项卡中。

**添加对象到工具箱中**

- 在 Tools 菜单上，单击 Customize Toolbox（定制工具箱）命令。
  或者
  将已经复制到剪贴板上的项目粘贴到工具箱中。
  或者
  从 Text 编辑器中将选择的代码段或文本拖动到工具箱中，然后将它们拖动到其他所要使用的文件或窗口中。

  **注意：** 只有扩展 com.ms.wfc.core.Component，并具有 ShowInToolbox 属性设置（该属性默认设置为真）的组件才显示。

要组织工具，用户可以：

- 通过按下 CTRL 键并拖动所选项到新选项卡的方法，来从一个选项卡

中复制工具箱项到另外的一个选项卡中。

- 通过拖动所选项到新选项卡的方法，来从一个选项卡中移动工具箱项到另外的一个选项卡中。

  **注意：**如果试着移动一个不可移动的项，将其拖动到新的选项卡中的操作将是复制该项，而不是移动它。

可以通过下列方法来添加和定制 Toolbox 选项卡：

- 单击在右击工具箱时所出现的快捷菜单上的 Add Tab（添加选项卡）命令来为定制的组创建选项卡。
- 通过拖动项目或选项卡到所需要的位置上来重新排列项目和选项卡。
- 单击选项卡标签并输入新的项的名称说明文本，或使用在快捷菜单上的 Rename Item（重命名项）命令。

**按列表方式查看工具箱对象**

- 右击工具箱来显示快捷菜单，并且单击其中的 List View（列表显示）命令。则以列表方式显示工具箱中有效选项卡中的内容。该列表包含了与有效 Toolbox 选项卡相关联的图标和描述信息。

**按图标方式查看 Toolbox 对象**

- 右击工具箱来显示快捷菜单，并且清除 List View 复选框。这仅以图标方式显示工具箱中有效选项卡的内容。

**从工具箱中删除对象**

- 选择要删除的项目，并且按下 DELETE 键。

**恢复被删除的控件**

1. 右击工具箱来显示快捷菜单，然后单击 Customize Toolbox（定制工具箱）。
2. 在 Customize Toolbox 对话框中，单击 WFC Controls 选项卡。
3. 在 WFC Controls 选项卡中，单击所要添加的控件的复选框。

## 添加控件到窗体中

用户可以通过在工具箱中选择控件，来在设计模式中将控件添加到窗体中。

**添加控件**

- 在工具箱中，单击想要添加窗体的控件，并且将其拖动到窗体上。该控件在窗体上以默认大小显示。
  或
  在工具箱中，双击想要添加窗体的控件会将其添加到窗体的角上。该控件在窗体上以默认大小显示。可以将其拖动到要求的位置。

  **注意：**在设计时将控件放入窗体时，只添加用于该控件的类名。用于该控件的代码只有在关闭窗体时才生成。如果在工具箱中编辑已有的控件，用户必须关闭并重新打开该窗体或重新加载项目，使窗体选取控件的更新类文件。

**调整控件大小**

- 选择所要调整大小的控件，然后使用大小控制柄来调整大小。

  **注意：**还可以使用 SHIFT + 方向键来调整控件的大小。

**添加事件处理程序模板到控件中**

- 双击窗体或控件为这些窗体或控件的默认事件生成方法。该事件处理程序方法在 Form 类中创建。

  **注意：**还可以通过 Properties 窗口中的事件视图来添加事件处理程序。

## 添加事件处理程序

可以使用 Forms Designer 来创建事件处理程序。事件处理程序是在代码中的一段程序，用来确定事件发生时将执行的动作，如用户单击某个按钮时。

例如，用于 mouseDown 事件的事件处理程序提供了 mouseEvent 对象，它允许用户检测鼠标的哪个按钮被按下，鼠标在窗体中的位置，以及单击时按下了键盘上的哪个键。

可以通过在 Properties 窗口中的事件视图来添加事件处理程序。用于默认事件的处理程序还可以通过双击来添加。此外，如果事件的信号相同，还可以指定一个已有的处理程序到其他控件的事件中。

### 使用 Properties 窗口的事件视图来添加事件处理程序

1. 单击要创建事件处理程序的控件（如一个按钮控件）或窗体。
2. 在 Properties 窗口中，单击事件视图按钮（由一个闪电形状符号来标识）。
3. 在可用的事件列表中，单击某个事件（例如 click）。
4. 在事件名称右边的框中，输入事件的名称（例如 MyButton_click），然后按下 ENTER 键。

   代码中生成代码模板，与下面的内容类似：

```
private void MyButton_click(Object sender , Event e) {
}
```

   在该例中，sender 是事件的源，而 e 是提供有关事件信息的 Event 对象。

### 为默认事件添加处理程序

- 双击窗体或控件为默认窗体或控件事件生成处理程序方法。

  或

  在 Properties 窗口中的事件视图中，双击事件来为默认的事件创建处理程序。

### 为多控件共享的事件使用同样的处理程序

1. 在 Properties 窗口中的事件视图中，单击用于第一个控件的事件。
2. 在事件名称右边的框中，输入处理程序的名称并按 ENTER 键。
3. 在该窗体上，选择第二个控件。

4.在 Properties 窗口中，为第二个控件单击同样的事件。
5.单击向下箭头来查看该事件类型已有的处理程序的列表。
6.选择事件处理程序的名称。只有已有的处理程序被使用。在代码中没有创建新的模板。

## 格式化窗体

可以使用窗体的位置属性、Tools 菜单上的网格线选项和 Format 菜单上的各种选项来格式化放在窗体上的各种对象。

### 改变控件的位置属性

1.单击所要改变的控件。
2.在 View 菜单上，单击 Properties Window（属性窗口）命令。
3.在 Properties 窗口中，双击 location（位置）属性来展开该节点。
4.为该控件输入 x 和 y 坐标。

**注意：**用户还可以在程序中通过设置 x 和 y 坐标来改变控件的位置。

### 设置网格选项

1.在 Tools 菜单上，单击 Options（选项）命令。
2.在 Options 对话框中项目的树形视图中，双击 Forms Designer，然后单击 WFC。

用户还可以使用该对话框来设置是否显示网格线、以像素为单位显示网

格的高度和宽度，以及到网格的对齐方式等默认值。
要在窗体中格式化控件，用户可以使用 Format 菜单中的选项来：

- 排列控件。
- 使多个控件的大小一致。
- 设置网格大小。
- 调整水平和垂直间距。
- 调整前后顺序。
- 在某个位置锁定控件。

**在窗体中排列两个控件**

1. 单击第一个控件（该控件成为参考标准），然后按下 CTRL 键并单击第二个控件。
2. 在 Format 菜单上，指向 Align（排列），然后单击 Lefts（左对齐）。第二个控件按照第一个控件的左边界对齐。

   **注意：** 控件能够在其顶部叠放。

## 修改窗体布局行为

使用 Forms Designer，用户可以定义放在窗体上控件的行为。例如，用户使用 anchor（锚点）属性可以将一个选项卡控件定位在窗体的顶部，或者可以修改窗体的停靠行为。

**在窗体中锚定控件**

1. 在窗体中放置控件。
2. 单击该控件使得该控件的属性出现在 Properties 窗口中。
3. 在 Properties 窗口中，双击 anchor 属性。
4. 在 Anchor 编辑器中，单击十字线来允许或禁止在控件和窗体边界之间锚定。

**修改停靠行为**

1. 在窗体中放置控件。
2. 单击控件，使控件的属性在 Properties 窗口中出现。
3. 在 Properties 窗口中，双击 dock（停靠）属性。
4. 在 Docking（停靠）编辑器中，单击条来允许或禁止控件停放。单击 None（无）来禁止停放。

**注意：** 控件将会自动改变大小以适应所停放边的边界。

多数控件能够停放到窗体的边界上或能够填充整个窗体。

## 使用 Properties窗口设置属性

属性是定义控件如何显示及在运行时如何运行的特征。每个窗体和每个控件都有一个关联的定制属性页。修改在该页（Properties 窗口）中的属性会影响在 initForm 方法中为该控件所生成的代码。控件属性包括控件的外观、行为和位置。而窗体的背景颜色、允许或禁止状态行、显示状态行、字体和默认大小和位置等都可以在窗体的 Properties 窗口中设置。

**编辑控件或窗体的属性**

1. 在窗体上，单击所要编辑的控件；如果要编辑该窗体，则单击该窗体。与选中的控件关联的属性页显示在 Properties 窗口中。
2. 在 Properties 窗口中，选择该控件中所要修改的属性。例如，要改变按钮控件上名称的字体风格，则选择 font 属性。

   **注意：** 如果选择了多个控件，定制的属性页中将只显示它们公共的属性。

使用公共对话框和编辑器的复杂属性有：

- 文本输入——大小和字符串（适当位置字符串的编辑更改会实时出现）。
- 组合框选择——光标、枚举符和布尔值。
- 字体选取器——用于选择字体特征的公共对话框。
- 颜色选取器——用于颜色选择的调色板的公共对话框。
- 列表项——用于创建初始列表项的窗口。
- Anchor 和 Dock 编辑器——用于设置位置行为。

单击复合属性的节点可以访问到这些子属性。

## 为窗体创建菜单

用户可以由 Forms Designer 所提供的菜单控件来创建主菜单和上下文关联菜单。

# 添加和修改菜单

如果想要应用程序在窗体中提供给用户一个命令集，菜单提供了方便而一致的组织命令的方法，并且用户可以方便地访问这些命令。菜单栏出现在窗体的标题栏下方，包含一个和多个菜单名。当单击一个菜单名时（如 File），会出现包含菜单项的列表。菜单项中可以包含命令（如 New 和 Exit）、分隔线和子菜单名。用户所见到的每个菜单项分别对应着一个定义的菜单控件属性。在一个窗体中能够添加多个菜单。

## 在窗体中添加菜单

1. 在工具箱中的 WFC Controls（WFC 控件）下，双击 MainMenu 控件。

   **注意：** 用户还可以选中 MainMenu 控件并将其拖动到窗体上。

   MainMenu 控件插入到窗体上，并且显示出菜单栏上第一项的默认位置。两个可见的标记（Type Here）直接出现在该位置的右侧和其下方来指示下一个可用的编辑位置。

   **注意：** 该窗体的 MainMenu 属性默认设置为 MainMenu1。

2. 在默认位置文本框中输入第一个菜单标题名（例如，输入 File 来添加到 File 菜单）。

   菜单标题添加到菜单中，并且默认位置文本框移动到下一个菜单标题位置。继续输入菜单标题，直到所有的标题输入完毕为止。

窗体可以有几个根据应用程序的状态来改变的菜单项，如 File/Save 和

File/Save As。使用属性可以将每个菜单项设置为选择或禁止选择、选中或非选中及可见或非可见等。

如果想要改变窗体所使用的主菜单，可以将该窗体的 menu 属性改变为不同的菜单名称，如 mainMenu2。

用户可以将访问键添加到菜单中。访问键为用户提供了只使用键盘来访问菜单的方法。访问键相当于在菜单上的带下划线的字母。

**添加访问键到菜单中**

1. 在想要添加访问键的菜单标题上单击。
2. 将光标放在标题框中想要作为访问键的字母前面，并且输入一个&符号。例如，在 File 菜单上，将光标放在 File 中的“F”前面，然后输入一个&符号。还可以通过 Text 属性来编辑菜单项的标题。

快捷键增加了菜单的可访问性。快捷键是一个键组合，如用于复制的 CTRL+C，该键组合调用特定的命令。

**添加快捷键到菜单中**

1. 单击在所要添加快捷键的菜单项标题的右侧，出现一个下拉列表。
2. 从下拉列表中选择一个键组合。

   或

   在 Properties 窗口中，单击 shortcut（快捷键）属性并选择一个键组合。

**删除快捷键**

1. 选择菜单项。

2.将 shortcut 属性设置为 none（无）。

偶尔，可能想提供给用户各种命令，这样就可以在应用程序中使用它们。例如，在应用程序中可能有几种可用的工具栏。用户可以通过单击菜单上的工具栏名称的方法来选择显示哪些工具栏，同时，在菜单上显示的工具栏名称的旁边出现选择标记。

**在菜单上添加复选标记**

1.选中一个菜单项。
2.单击紧靠菜单名左边的位置，则出现一个复选框。
或者
在 Properties 窗口中，单击 checked（选中的）属性。

**移动菜单**

1.选中需要的菜单项标题。
2.拖动该菜单标题到新的位置。
菜单标题移动并对齐之后，受到该变化影响的其他菜单标题均适当移动。

**注意：**移动一个最高级别的菜单标题时，其所有的子菜单同时移动。在移动过程中子菜单并不显示。

**编辑菜单标题**

1.单击所要编辑的菜单标题。
2.在包含菜单标题的文本框中进行必要的改变。

在菜单中，分隔条用来在两组命令中添加一条明显的线。

**创建菜单分隔条**

1.单击所要添加的分隔条上面的菜单标题。
2.按下 HYPHEN(-)键并按 ENTER 键。
一个分隔条立即出现在菜单标题的下方。
或者
右击鼠标按钮，然后在出现的上下文关联菜单中选择 Insert Separator（插入分隔条）命令。

分隔条可以使用操作菜单标题的同样的方法来进行移动和删除。

**从窗体中删除菜单项**

- 右击所要删除的菜单项标题，并且从上下文关联菜单中单击 Delete（删除）命令。

  **注意：**要删除整个菜单，选择每个菜单标题并按下 DELETE 键，或者单击所要删除的 MainMenu 控件，并按下 DELETE 键。

只有删除带有子菜单的菜单标题时才出现确认删除对话框。

**编辑与菜单项关联的名称**

1.在菜单项上右击。
2.单击 Edit Names（编辑名称）来进到该菜单的 Edit Names 模式。
3.在所要编辑的与菜单项关联的名称上单击（如 menuItem1）。

## 上下文关联菜单

上下文关联菜单能够与窗体中的控件相关联。当用户右击控件时这些菜单出现。

**添加窗体的上下文关联菜单**

1. 在工具箱中的 WFC Controls 下，双击 ContextMenu 控件。

   **注意：**使用任何一种方法在窗体中添加控件之后，用户同样可以在工具箱中选择 ContextMenu 控件并将该控件拖动到窗体之上。

2. 双击 ContextMenu 控件进入到设计模式。

**指定控件的上下文关联菜单**

1. 单击要指定上下文关联菜单的控件。
2. 在 Properties 窗口中，将上下文关联菜单的名称添加到 contextMenu 属性中。第一个上下文关联菜单的名称默认为 contextMenu1。

可以用编辑 MainMenu 控件的方法来编辑上下文关联菜单。更多信息参见本章前面的“添加和编辑菜单”一节。

## 在窗体控件中添加工具提示

工具提示是一个文本串，当鼠标指针“盘旋”在控件上时显示该提示。在 Visual J++中，用一个单独的 WFC 控件来实现窗体控件的工具提示功

能。时间、延迟和其他属性可以使用在窗体上的每个控件的属性页来设置。

**添加控件的工具提示**

1. 在 WFC 工具箱中，将 ToolTip 控件添加到窗体。
2. 编辑每个控件的属性来添加说明文本。

## 代码生成

Visual J++中代码的改变和窗体视图（代码由可视布局生成）这两个方面是同步的。有关正在修改的方法（或类）的信息被格式化到源代码中。同样，对源代码的修改也送到 Forms Designer 中。

这种双向编辑方法不能同时编辑源代码和可视布局。用于创建可视布局的代码在 Forms Designer 使用时是禁用的。当 Forms Designer 窗口关闭和窗体保存时，布局作为代码存储。当 Forms Designer 窗口打开时，整个 initForm 功能被锁定。

在未单击 File 菜单上的 Save（保存）命令，或未关闭 Forms 编辑器时，Forms 编辑器中编辑的代码是不连续的。

用户可以按照下面列出的句法来使用 Forms Designer 来编辑类：

```
class isEditable extends [UserControl , Form , Design Page ] {
    container components = new container (this);
    private void initForm() {
```

[ *form initialization code goes here* ]

}

}

初始化代码必须按照规定的语法分析规则：

- 初始化代码必须在 initForm 函数中。
- 不分析条件代码（没有 if，while 或 switch 语句）。

# 第 3 章　编辑代码

Visual J++提供了几种用来创建、修改和管理项目代码的方法。本章的主题和过程将提供给用户第一手经验，用来帮助用户研究 Text 编辑器、Class Outline（类大纲视图），WFC Component Builder（WFC 成员建立器）和 Object Browser（对象浏览器）。

## 使用Text编辑器管理文件

程序开发环境包括一个完整的 Text 编辑器来管理、编辑和打印源文件。如果用户已经使用过其他基于 Windows 的文本编辑器，那么，使用这个编辑器的过程与使用那些文本编辑器很类似。另外，Visual J++还用几种节省时间的新特性来增强了这个 Text 编辑器，如语句补全、动态语法检查和在 Text 编辑器和 Class Outline 之间的双向作用等——这些只是 IntelliSense 提供给 Visual J++开发者几种服务。

## 在Text编辑器中拆分窗口

**拆分 Text 编辑器的窗口**

1. 将鼠标指针指向垂直滚动条顶部的拆分框。

2. 当鼠标指针变为改变窗口大小光标时，拖动拆分条到所需要的位置。

**返回到单一窗口**

- 双击拆分条。

**在长文档的不同位置移动或复制文本**

1. 将窗口拆分为两个窗格。
2. 在其中的一个窗格中显示想要移动或复制的文本。
3. 在另一个窗格内显示这些代码的目的地。
4. 选择并拖动这些代码穿过拆分条到目标窗格。

## 在全屏模式中查看文件

**开始或结束全屏模式：**

- 在 View（视图）菜单中，单击 Full Screen（全屏显示）。

## 在单一文件中查找和替换文本

**在单一文件中查找文本**

1.将插入点移动到文件中想要开始搜索的位置。
2.在 Edit 菜单中，单击 Find and Replace（查找与替换）。
出现 Find（查找）工具窗口。

3.在 Find 按钮附近的文本框中，输入要查询的文本。
4.从 Look In 组合框中，选择 Current Document（当前文档）。
5.单击 Find（查找）开始搜索
这将突出显示首次出现的搜索项。
6.要继续查找，单击 Find 按钮。

## 在单一文件中替换文本

1.将插入点移动到文件中想要开始搜索的位置。
2.在 Edit 菜单中，单击 Find and Replace（查找与替换）。
出现 Find 工具窗口。
3.在 Find 按钮附近的文本框中，输入要查询的文本。
用户也可以使用下拉列表来从以前搜索的字符串列表中选择。
4.从 Look In 组合框中，选择 Current Document。
5.在 Replace（替换）按钮旁边的文本框中，输入替换的文本。
6.单击 Replace。
在 Text 编辑器中首次出现的匹配字符将突出显示。

> **注意：**要替换在该文件中所有的匹配字符，单击 Replace All（全部替换）。

7.再次单击 Replace。
这将替换当前选中的部分，并突出显示搜索到的下一个匹配项。

# 在多个文件中查找文本

## 在多个文件中查找文本

1. 在 Edit 菜单中，单击 Find and Replace（查找与替换）。
   出现 Find 工具窗口。
2. 在 Find 按钮附近的文本框中，输入要查询的文本。
   也可以使用下拉列表来从以前搜索的字符串列表中选择。
3. 从 Look In 组合框中，选择要搜索的文件类型。
4. 单击 Browse（浏览），显示 Look In 对话框。
5. 突出显示要开始搜索的最上层文件夹。
6. 单击 Add（添加）按钮。
   所选中的文件夹名称出现在 Look In 对话框中的底部文本框中。
7. 单击 OK 返回到 Find 对话框。
8. 单击 subfolders（查找子文件夹）复选框，用以包含在前面所选择的最高层文件夹中的子目录中的文件。
9. 单击 Find 开始搜索。
   在 Find 对话框的底部窗口中出现包含搜索文本的文件列表。
10. 在列表中的某个文件名上双击，可以进入该文件。
    包含该文件的编辑器窗口打开，并显示选中的搜索文本行。

## 查找匹配分界符

源代码经常成组使用分界符，如()，{}和<>。这些组称为级别。编辑器了解嵌套的级别，并匹配正确的分界符，即使这些级别横跨过了几页，并在其中包含了多个级别。利用编辑器的这种功能，用户可以在级别的开始及结束位置之间快速跳转。

### 查找匹配分界符

1. 将插入点移动到分界符的前面或后面。
2. 按下键盘快捷键 CTRL+]。
   插入点向前或向后移动到匹配的分界符上。再次选择该命令将插入点返回到开始位置上。如果没有找到匹配的分界符，则插入点不会移动。

## 使用书签导航

在源文件中可以设置书签来标记经常访问的行。一旦设置了书签，就可以使用菜单或键盘命令移动到该书签的位置。当不再需要书签时，用户可以将其删除。

### 设置书签

1. 将插入点移动到要设置书签的行。
2. 在 Edit 菜单上，单击 Bookmarks（书签），并在层叠菜单中选择 Toggle Bookmarks（触发书签）。

或

按下 Toggle Bookmark 的键盘快捷键，CTRL+K，CTRL+K。

如果允许编辑器显示书签的话，则在挨着选择行的页边上出现一个浅蓝色的矩形。

**导航到下一个书签**

- 在 Edit 菜单上，单击 Bookmarks，然后在层叠菜单中选择 Next Bookmark（下一个书签）命令。

  或者

  按下 Next Bookmark（下一个书签）的快捷键，CTRL+K，CTRL+N。

**导航到上一个书签**

- 在 Edit 菜单上，单击 Bookmarks，然后在层叠菜单中选择 Previous Bookmark（上一个书签）命令。

  或者

  按下 Previous Bookmark（上一个书签）的快捷键，CTRL+K，CTRL+P。

  一个浅蓝色的矩形从页边消失。

**删除书签**

1. 将插入点移动到一个已有书签的行。
2. 在 Edit 菜单中，单击 Bookmarks，然后在层叠菜单中选择 Toggle Bookmark。

   或者

按下 Toggle Bookmark 的键盘快捷键，CTRL+K，CTRL+K。

**删除所有的书签**

- 在 Edit 菜单上，单击 Bookmarks，然后在层叠菜单中选择 Clear All Bookmarks（清除所有书签）命令。

  或者

  按下 Clear All Bookmarks（清除所有书签）的快捷键，CTRL+K，CTRL+L。

## 选择文本框

用户可以选择文本框，也就是我们知道的文本块，来执行剪切、复制、删除、增加或减少缩进量等。

**启用文本框选择**

1. 在 Tools（工具）菜单中，单击 Options（选项），显示 Options 对话框。
2. 在 Options 对话框中展开树形视图中的 Text Editor（文本编辑器）节点。
3. 展开 Per Language 节点并选择 Java。
4. 在选项的 Settings（设置）组中，选择 Box Selection（矩形选择）单选钮。

**选择文本块**

1. 将鼠标指针指向要开始选择的文本。
2. 按下鼠标左键并移动鼠标来突出显示文本块。

当放开鼠标左键时，一个文本块被选中，并且可以用于剪切、复制、删除及改变缩进量选项。

**注意：**单击鼠标左键来取消已选择的文本块。

## 使用语句补全功能编写代码

Visual J++的语句补全功能为用户查看以前的工作，以节省用户的时间。语句补全自动显示成员列表和参数信息，以给用户提供类、成员和方法特征的列表，这些均与当前.java 源文件上下文密切相关。
为了介绍语句补全特性，该方案将显示如何使用 java.lang.String 和 java.lang.System 类来建立语句。当建立项目并运行应用程序之后，程序将在由 JVIEW 提供的控制台窗口中显示一条消息。该方案包含下列过程：

- 在 Text Editor（文本编辑器）中启用/禁用 Statement Completion Options（语句补全功能选项）。
- 使用 Word Completion（单词补全）功能创建语句。
- 选择 Overload Method（重载方法）。
- 从 Member List（成员列表）中选择方法。
- 使用 Parameter Info（参数信息）建立 Argument List（参数列表）。

**注意：**此方案是用 Visual J++ Console Application（Visual J++控制台应用程序）模板来创建的。在任意 Visual J++项目内，使用

任意 .java 源代码来完成下面的过程。如果想创建控制台应用程序来完成此方案，请参阅第一章中的“创建控制台应用程序”，并且在过程之前按照下列步骤来创建并打开项目。

## 在 Text 编辑器中启用/禁用语句补全选项

在开发环境中，用户可以启用或禁用语句补全的 Auto list members（自动成员列表）和 Parameter information（参数信息）选项。

### 启用 Statement Completion 选项

1. 在 Tool 菜单上，单击 Options 命令来显示 Options 对话框。
2. 从 Options 对话框最左边窗格中的树形视图中，选择在 Text Editor 和 Per Language 节点旁边的加号（+）来展开这些节点。
3. 选择 Java 来显示 Text Editor/Per Language/Java 属性页。
4. 在该属性页的 Statement Completion 组中，单击 Auto list members 和 Parameter information 复选框来启用这些选项。

Visual J++ 默认启用语句补全的 Auto list members 和 Parameter information。

**注意：** 清除对应于 Auto list members 或 Parameter information 的复选框以便它们可以禁用各自的选项。

## 使用单词补全创建语句

例如，单词补全特征在用户输入 String 和 System 类名称时提供可见的线索，并且创建一个新的 String 对象。单词补全可用于任何 Visual J++项目。

**注意：** 在下面列出的方案中，代码示例是使用 Visual J++ Console Application 项目创建的。如果需要创建 Console Application 项目来完成该项目，见本书第一章中的“创建控制台应用程序”一节。并且在过程之前按照下列的步骤创建并打开项目。

**通过单词补全特性提供的列表中选择项目**

1. 在 View 菜单上，单击 Project Explorer 来打开 Project Explorer。
2. 在 Project Explorer 中，单击项目名称左边的加号（+）来展开项目。
3. 突出显示包含项目的 main()方法（默认为 Class1.java），并且从快捷菜单中选择 View Code（查看代码）命令。
   Visual J++打开 Text 编辑器并且加载所选择的.java 文件。该文件现在已经做好了编辑的准备。
4. 在应用程序的 main()方法的大括号之间输入字母“S”，并且将光标留在该字符的右边。

**提示：** 注意在字母下面的红色波浪线。因为字母 S 在程序的上下文中没有意义，所以 IntelliSense 给出了一个可见的线索来提示句法

错误。有关可见编辑线索的更多信息，见本章后面的“使用动态句法检查来查找错误”一节。

5. 在 Edit 菜单上，单击 Complete Word 来显示一个类和其他 IntelliSense 识别元素的列表框。

   **提示：**可以使用键盘快捷键 CTRL+SPACE，而不是用 Edit 菜单，选择 Complete Word 选项来显示该列表框。

6. 在列表框始终显示的情况下，在.java 文件中建立的语句“S”的后面，接着输入字母“T”。

   Statement Completion 现在在它的列表中选中 StackOverFlowError，这是在该程序的上下文中第一个以“ST”开头的元素。

7. 现在在语句中的“ST”后面输入“R”，Statement Completion 在列表框中将突出显示条移动到“String”。

8. 按下 Tab 键将 String 插入到所建立的语句中。

   按下 Tab 键或其他任何一个非字母或数字键，如句号或括号，将所选择的项目放入到语句中插入点所在的位置。

   **提示：**用户可以使用向上或向下箭头突出显示所需要的选择代替输入字符，来让语句补全查找所需要的元素。

### 在 new 运算符后选择类名称

1. 在 String 后面输入下列的代码来继续建立语句。

```
[String] myStr = new
```

2. 在 new 后面输入空格，语句补全自动显示在程序范围中可用的类名称列表。
3. 输入“STR”，或“String”字符中足够的字母，来在该列表中选择 String 项目。
4. 按下 Tab 键，String 插入到 New 之后。
   所建立的语句应该像如下所示：

```
String myStr = new String
```

要继续建立显示一行文本的语句，需要一个用于新的 String 对象的构造器。使用语句补全的 Parameter Info 选项选择 String 构造器的过程，见下一节，“选择重载方法”。

如果知道怎样部分地拼写类或项目名称，用下述过程来加速拼写、选择正确的拼写与大小写的处理。

**使用单词补全特性完成输入单字**

1. 在 .java 文件中，将光标移动到一个新行，并且输入“SY”。
2. 按下 CTRL+SPACE（单词补全的键盘快捷键）。
   语句补全完成该单字，System，如果在文件中 System 是唯一以“SY”开头的项目，则语句补全还将其插入到代码中。如果有多个项目以“SY”开始，语句补全将显示一个列表框。

## 选择重载方法

本例使用语句补全来帮助用户选择用于 String 类的重载构造器方法。

> **注意：**如果用户已经完成了“使用 Word Completion 创建语句”一节中的步骤，那么就可以继续完成下面的过程。如果没有完成这些步骤，则在为 String 对象选择一个重载构造器方法之前，需要花费几分钟的时间来完成这些过程。

在开始这些过程之前，需要确定已经将包含下面列出的代码的.java 文件加载到 Text 编辑器。

```
String mystr = new String
```

### 为类选择一个重载方法

1. 将插入点放到在语句中的最后一个 String 后面，然后输入一个左圆括号“(”。

   一个弹出式窗口出现，显示用于 String 类的某个构造器方法的特征。在该窗口左边的旋转控件指出有多少用于 String 类的重载构造器方法（11）是有效的，和正在显示哪一个（1）。

   > **注意：**对于那些没有重载的方法和构造器，该列表框将被 Parameter Info 文本框所替代。使用 Parameter Info 特性的示例可参见使用 Parameter Info 建立参数列表一部分。

2. 在旋转控件的任意部分单击来显示其他 String 构造器方法的特征，直

到找到一个带有 String 参数的特征为止。
在本例中，构造器成员 11 满足该条件。

3. 在左圆括号之后，输入"Hellow World !");。
完成的语句应该如下所示：

```
String myStr = new String ("Hellow World !");
```

语句补全还显示类的公共方法和字段。要显示用于 String 对象的成员列表，见下一节“从成员列表中选择方法”。

## 从成员列表中选择方法

本例使用语句补全的成员列表特性来帮助用户选择 String 类的方法。用户也许使用这些过程在 Member List 中选择类成员变量。

> **注意：**如果用户已经完成了使用 Word Completion 创建语句和选择重载方法中的步骤，那么就可以继续完成下面的过程。如果没有完成这些步骤，则在为 String 对象选择一个方法之前，需要花费几分钟的时间来完成这些过程。

在开始这些过程之前，需要确定已经将包含下面列出代码的.java 文件加载到 Text 编辑器。

```
String myStr = new String ("Hello World !");
```

### 从 Member List 中选择类方法

1. 在新行上，输入：

```
if (mystr.
```

2. 当在 mystr 后输入“.”（圆点符号）时，语句补全显示属于该 String 类的方法的列表。

   **提示：**如果输入圆点符号之后没有得到成员列表，用户可以从 Edit 菜单中选择 List Members（列出成员）命令，或者使用键盘快捷键 CTRL+J。如果这种情况连续发生，则需要确定一下是否启用了语句补全特性。

3. 输入“equalsI”来在 Members List 中突出显示 String 类的 equalsIgnoreCase 方法。

4. 键入 Tab 键将 equalsIgnoreCase 方法插入到 myStr.的圆点符号后面。在代码中的这两行应该如下所示：

```
String myStr = new String ("Hello World!");
if (myStr.equalsIgnoreCase
```

要继续使用语句补全的 parameter Info 特性建立下一个语句，见使用 Parameter Info 建立参数列表一部分。

## 使用参数信息建立参数列表

本例使用语句补全的参数信息特性来显示用于方法的参数信息。

**注意：**如果用户已经完成了“使用 Word Completion 创建语句”、“选择重载方法”和“从 Member List 中选择方法”中的步骤，那么就可以继续完成下面的过程。如果没有完成这些步骤，则在为 String 对象选择一个方法之前，需要花费几分钟的时间来完成这些过程。

在开始这些过程之前，需要确定已经将包含下面列出的代码的.java 文件加载到 Text 编辑器。

```
String myStr = new String ("Hello World !");
if (myStr.equalsIgnoreCase
```

**使用 Parameter Info 选择建立参数列表**

1. 继续建立 if 语句，在 myStr.equalsIgnoreCase 后面紧接着输入左圆括号“(”。

   Parameter Info 使用单独的参数（String p1）以粗体显示方法的声明。

   **注意：**在此时，方法只有一个参数（String p1）。如果选择了带有多个参数的方法，IntelliSense 显示参数的类型和在方法声明中的位置。当在方法名称后面输入左圆括号时，IntelliSense 以粗体显示第一个参数。当在两个参数之间添加逗号时，IntelliSense 以粗体显示下一个需要用来完成方法的调用参数类型和位置信息。

   **提示：**如果当输入左圆括号之后没有得到带有参数信息的弹出窗口，用户可以从 Edit 菜单中选择 Parameter Info（参数信息）命令，

或者使用键盘快捷键 CTRL+SHIFT+I。如果这种情况连续发生，则需要确定一下是否启用了语句补全特性。

2. 在左圆括号后，通过输入：

```
"hello world!"))
```

来完成句子。
完成的语句应该如下所示：

```
if(myStr.equalsIgnoreCase( "hello world!"))
```

3. 使用语句补全的 Word Completion、Member List 和 Parameter Info 选项，并通过在 if 语句的右括号之后添加下列代码来完成语句：

```
  {
  System.out.println("The strings are the same. ");
  }
else
  {
  system.out.println("The strings are different.")
  }
return;
```

4. 建立程序。
5. 在 Text 编辑器中，将光标放在“return”上（程序代码的最后一行语句）。单击鼠标右键并从快捷菜单中选择 Run To Cursor（运行到光标）。

6. 在 JVIEW 的控制台窗口中查看下列结果：

The strings are the same.

## 使用动态语法检查查找错误

当用户在 Text 编辑器中编写代码时，Visual J++还提供动态语法检查来帮助用户。除了语句补全所提供的这些信息之外，在建立程序的语句时，还可以在窗体中看到红色波浪线的可见线索和错误提示。

当在.java 文件中开始输入时，红色波浪线出现在代码元素，如类名称、成员名称和符号的下面。当看见红色波浪线时，说明 Intellisense 正在提示当前输入的代码有语法上的错误。而当继续输入并完成语句输入后，红色的波浪线有可能消失，这取决于是否正确地完成语句。

对于使用红色波浪线标记的每个语法错误，与该语法错误关联的任务将出现在 Task list 中。这就提供了在编译程序之前需要修改的项目列表。

### 从红色波浪线中获得错误提示和错误帮助

1.将光标停在红色波浪线上。

IntelliSense 显示最适合源代码上下文关联的 Error Tip（错误提示）。有时，该提示将仅仅是“Syntax Error（语法错误）”。

**提示：**如果在明显的句法错误下没有看到红色波浪线，需要证实一下是否启用了支持动态语法检查的选项。

2. 当把光标放在红色波浪线上时，右击鼠标出现快捷菜单。
3. 在快捷菜单中，选择 Error Help（错误帮助）命令，则用户可以得到红色波浪线标识的错误的在线帮助。

对于演示红色波浪线的使用，见本章前面的“使用 Word Completion 创建语句”一节。

## 在 Text编辑器中启用/禁用动态语法检查

在开发环境中，用户可以启用或禁用编辑器的动态语法检查选项。

### 启用动态语法检查选项

1. 在 Tools 菜单上，单击 Options 来显示 Options 对话框。
2. 从 Options 对话框中最左边的属性视图中，单击在 Text Editor 节点旁边的加号（“+”）来展开该节点。
3. 选择 Java Tasks 来显示 Tasks（任务）和 Error Display（错误显示）的属性页。
4. 在属性页的 Tasks 组中，单击 Check syntax as you type（在输入时检查语法）对应的复选框。
5. 在属性页的 Error Display 组中，单击 Underline syntax errors as you type（在输入时使用下划线标识错误）对应的复选框。

Visual J++默认启用这些动态语法检查选项。

注意：清除 Check syntax as you type 或 Underline syntax errors as you type 所对应的复选框会禁用相应的选项。

## 从 Text编辑器中更新类大纲

只要用户从 Text 编辑器中添加方法、成员变量或类到一个已有的.java 源文件中，Visual J++就使用 IntelliSense 来动态更新文件的 Class Outline 树视图。如果想要将 Javadoc 注释添加到代码中，IntelliSense 将创建一个注释块，并且在 Class Outline 的 Javadoc 格中显示注释。

## 从 Text编辑器中添加项目到 Class Outline中

尽管 Visual J++提供 Class Outline 和 WFC Component Builder 用来帮助用户将方法、成员变量、属性、事件和类添加到项目中，但用户还可以使用从 Text 编辑器中将代码直接输入到.java 文件中的方法来将其添加到项目中。当用户将新类和类成员的代码添加到.java 文件中时，IntelliSense 立即在项目的 Class Outline 中为这个新的类或类成员显示适当的图标。下列过程演示了当新的类和方法从 Text 编辑器中添加到源文件时，Class Outline 的动态更新过程。

注意：在本节和下一节的代码示例，是使用 Visual J++ Console Application 项目创建的。用户也可以使用任何已有的 Visual J++ 项目来重新演示一下这些方法的结果。如果想要为下面的过程创建

一个 Console Application 项目，见“创建控制台应用程序”一节，并且在执行下列步骤之前创建并打开该项目。

**从 Text 编辑器中将新类添加到 Class Outline**

1. 在 Project Explorer 中，单击项目名称左边的加号（“+”）来展开项目。
2. 双击包含项目的 main()方法的.java 文件的图标或文件名（默认为 Class1.java）。

   Visual J++打开 Text 编辑器并且加载该.java 文件。该文件现已做好被编辑的准备。
3. 在 View 菜单上，单击 Other Windows（其他窗口），并从出现的层叠菜单中选择 Document Outline（文档大纲）命令。

   带有文件的折叠树形视图的 Class Outline 出现。
4. 从 Text 编辑器中，在用于 Class1 的类定义的结束大括号后面，将下面列出的代码添加到.java 源文件中：

```
class Greeting
{
}
```

5. 在 Class Outline 中，注意一个新的图标已经添加到项目文件的树形视图中，该图标对应于刚刚创建的 Greeting 类。

**注意**：当使用 Text 编辑器在源文件中移动插入点时，Class Outline 不会显示插入点所经过的定义。要使得 Class Outline 与源文件同

步，在 Text 编辑器中右击声明，并且在出现的快捷菜单中单击 Sync Class Outline（同步类大纲视图）。

**从 Text 编辑器中将新方法添加到 Class Outline**

1. 在 Class Outline 中，展开 Greeting 类来显示用于 Superclasses（超级类）和 Inherited Members（继承成员）的图标。
2. 在 Text 编辑器中，在 Greeting 开始的大括号之后添加下列代码：

```
public static String hello()
{
   String strGreet = new String("Hello World!");
   return strGreet;
}
```

3. 在 Class Outline 中，注意将对应于刚刚创建的 hello()方法的新方法图标添加到项目中。

**注意：**当使用 Text 编辑器在源文件中移动插入点时，Class Outline 不会显示插入点所经过的定义。要使得 Class Outline 与源文件同步，在 Text 编辑器中右击声明，并且在出现的快捷菜单中单击 Sync Class Outline（同步类大纲视图）。

4. 要测试 Greeting 类的 hello()方法，在应用程序的 main()方法中添加下

列代码：

```
System.out.println(Greeting.hello())
return;
```

5. 建立程序。
6. 在 Text 编辑器中，将光标放到“return”上（该程序代码的最后一条语句）。右击鼠标按钮，并且在快捷菜单中选择 Run To Cursor（运行到光标位置）。
7. 在 JVIEW 的控制台窗口中查看下列结果：

```
hello World!
```

要在新的 hello()方法中添加 Javadoc 注释，参见下一节“添加 Javadoc 注释到源文件中”。

## 添加 Javadoc注释到源文件

在 Class Outline 中，文件的树形视图下面是一个 Javadoc 窗格。如果类、方法或成员变量具有 Javadoc 注释，则注释出现在该窗格中。当添加一个 Javadoc 注释到代码中时，只要在 Class Outline 中突出显示类、方法或成员变量名，IntelliSense 将在 Javadoc 窗格中显示注释的第一句。
下面的过程解释了如何添加 Javadoc 注释到方法；用户也可以使用同样的步骤添加 Javadoc 注释到类和成员变量中。

**注意：**这些过程需要使用一个 Console Application 项目，并且将要添加到 Class Outline 更新项目中所创建的代码中。如果没有完成将 Greeting 类和 Hello() 方法添加到基础的 Console Application 的步骤，则在继续操作之前，需要花费几分钟的时间来执行前面一节“从 Text 编辑器中添加项目到 Class Outline 中”的步骤。

**添加 javadoc 注释到方法**

1. 在 Project Explorer 中，单击项目名称左边的加号（“+”）来展开项目。
2. 双击包含项目的 main() 方法的 .java 文件的图标或文件名（默认为 Class1.java）。

   Visual J++ 打开 Text 编辑器并且加载该 .java 文件。该文件现已做好被编辑的准备。
3. 在 View 菜单上，单击 Other Windows（其他窗口），并从出现的层叠菜单中选择 Document Outline（文档大纲）命令。

   带有文件的折叠树形视图的 Class Outline 出现。
4. 在 Class Outline 中，展开 Greeting 类来显示用于 Superclasses（超级类）、Inherited Members（继承成员）和 hello() 方法的图标。
5. 在 Text 编辑器中，在源文件中的 hello() 方法声明之上输入 Javadoc 注释的开始符号“/**”。

   IntelliSense 通过插入 Javadoc 注释的结束符号“*/”来创建一个 Javadoc 注释块。
6. 在 Javadoc 注释的开始符号后面，输入下面的文字：

```
/** the hello() method is a static method that takes
* no arguments and returns a String object to the
* calling method.  The value returned will always
* be "Hello World!".
* /
```

7. 在 Class Outline 中，突出显示 hello()方法。
   注意所添加注释的第一句出现在 Javadoc 窗格中。

当正在创建类、方法和成员变量的 Javadoc 注释时，用户也许想要或需要添加一个和多个 Javadoc 字段到注释中。IntelliSense 显示在 Visual J++中可用的 Javadoc 字段列表。下面的过程将添加一个 author 字段到上面创建的 Javadoc 注释中。

**添加 javadoc 字段到 javadoc 注释中**

1. 在 Project Explorer 中，单击项目名称左边的加号（“+”）来展开项目。
2. 双击包含项目的 main()方法的.java 文件的图标或文件名（默认为 Class1.java）。
   Visual J++打开 Text 编辑器并且加载该.java 文件。该文件现已做好被编辑的准备。
3. 在 hello()方法的 Javadoc 注释中，在最后的句子之后并且在 Javadoc 结束字符“*/”之前，输入@sign。
   IntelliSense 显示一个有效的 Javadoc 注释字段列表。
4. 在列表框中双击"author”。

单词“author”被插入到Javadoc注释中的@sign的后面。

5. 在@author后面输入名字。

完成的Javadoc注释应该类似如下：

```
/** the hello() method is a static method that takes
* no arguments and returns a String object to the
* calling method. The value returned will always
* be "Hello World!".
* @author Mary Doe
* /
```

## 使用Class Outline管理代码

Class Outline窗口列出所有在.java源文件中定义的类、接口和委托。所有被导入的类和包含在文件的软件包中的类也被列出。只要在Text编辑器中打开一个.java文件，Class Outline将自动显示该文件的信息。

对于每个类来说，Class Outline自动显示下面列出的细节：

- 超类，也就是在分层结构中的所有其他类。
- 从超类中继承的成员，也就是从分层结构中所有其他类中集成的成员。
- 类执行的任何接口。
- 在类中定义的任何初始化块。

- 包含在类中的任何嵌套或成员类。
- 在类中包含的任何嵌套接口。
- 由类定义的方法和成员变量。

在 Class Outline 中选中项目时，所有关联的 javadoc 注释显示在 Class Outline 的下面窗格中。

使用 Class Outline，用户可以：

- 定位到在源文件中的某个定义上。
- 在源文件中修改类声明。
- 添加新的声明到源文件中。
- 添加源文件中接口方法的声明。
- 在源文件中删除声明。
- 在源文件中移动或复制声明。
- 忽略从超类中继承的方法。
- 在方法中设置断点。

只要在 Text 编辑器中打开.java 文件，Class Outline 就可以使用，并且，它默认与工具箱是通过选项卡链接的。如果关闭了 Class Outline，还可以通过在 View 菜单中将鼠标指针指向 Other Windows，并且单击 Document Outline 来重新打开它。

## 更新 Class Outline

在多数情况下，由 Class Outline 显示的项目与其在源文件中关联的声明

自动同步；例如，如果在类中改变方法的名称，该方法名称也立即在 Class Outline 中更新。可是，如果改变一个继承方法的声明，用户必须手工更新 Class Outline。Class Outline 不会自动更新继承方法的列表。
右击 Class Outline 窗口，并且单击在快捷菜单上的 Refresh（更新）命令来更新 Class Outline。

## 定位到定义位置

用户可以使用 Class Outline 快速移动插入点到源文件中指定的定义上。如果该定义属于其他的.java 文件（如一个继承方法的定义），并且这个源文件在计算机上是有效的，那么该文件将打开，并且插入点将移动到该定义上。如果这个源文件无效，那么将打开 Object Browse（对象浏览器）。

### 定位到定义位置

1. 在 Class Outline 中，右击所要定位到的定义项目。
2. 在快捷菜单上，单击 Go to definition（转到定义）。
   或者
   在 Class Outline 中双击该项目。

**注意：**当使用 Text 编辑器在源文件中移动插入点时，Class Outline 不会显示插入点所经过的定义。要使得 Class Outline 与源文件同步，在 Text 编辑器中右击声明，并且在出现的快捷菜单中单击 Sync Class Outline（同步类大纲视图）。

# 修改类声明

使用 Class Outline 可以修改类的一般属性。

## 修改类声明

1. 在 Class Outline 中，右击所要修改的类名称。
2. 在快捷菜单上，单击 Class Properties（类属性）。
3. 从 Access（访问）下拉列表中选择一个访问修改器。嵌套（或内部）类能够声明为 public, protected 或 default（软件包）。非嵌套类能够声明为公共或默认。有关嵌套类的信息，见“Java Languate Specification”一书中的“内部类”部分。
4. 选择附加的修改符，如 abstract、final 或 static（只有嵌套类能够声明为 static）。
5. 要插入用于该类的 Javadoc 注释，在 Javadoc comment（Javadoc 注释）框中输入注释文本。
6. 如果想要将类作为 WFC 组件，选择 Include WFC Component Support（ClassInfo）（包含 WFC 组件支持）来插入一个 ClassInfo 类（该选项只可用于非嵌套类）。ClassInfo 用于描述该组件的属性和事件。
7. 要将类作为 COM 对象，选择 COM Class。选中该选项时，@com.register 指令插入，用来将该类注册为 COM 类，使它可以被其他支持 COM 的应用程序访问（COM Class 选项只可用于非嵌套类）。
8. 选择 MTS Support（MTS 支持）下面的 Enabled（启用），可以将 Microsoft Transaction Server（Microsoft 事务服务器）支持添加到类中（该选项

仅可用于非嵌套类有效）。注意，因为事务支持仅对 COM 类有效，所以，启用 MTS 支持也就选择了 COM Class 选项。然后，用户可以从下拉列表中选择下面列出的选项：

- Required：组件为它的工作请求 MTS 事务。
- RequiresNew：组件为它的工作请求新的 MTS 事务（尽管已有事务，但仍创建新的事务）。
- Supported：组件在操作时不注意是否由 MTS 提供事务。
- Notsupported：该组件只支持 MTS 事务或是 MTS API。

@com.register 和 @com.transaction 指令自动添加到类中。有关 Microsoft Transaction Server 的更多信息，见 Visual Studio SDK 平台在线文档的“Getting Started with Microsoft Transaction server”部分。

9.单击 OK 来应用这些改变。

**注意：**如果使用 Text 编辑器来编辑类声明，Class Outline 自动更新用以显示这些改变。

## 添加新声明

用户可以使用 Class Outline 向类中添加方法或成员变量，或向 .java 文件中添加新类。

**注意：**如果使用 Text 编辑器来添加声明，则该项目自动添加到 Class Outline。

**添加新方法**

1. 在 Class Outline 中，右击将定义其方法的类名称。在出现的快捷菜单中，单击 Add Method（添加方法）。
2. 在 Add Method（添加方法）对话框中，在 Method Name（方法名）框中输入该方法的名称。
3. 从 Return Type（返回类型）下拉列表中选择方法的返回类型，或者输入自己的返回类型。
4. 要添加方法参数，单击在 Parameters（参数）下面的省略号（...）按钮：
   - 在 Edit Parameter List（编辑参数列表）中，从 Type（类型）下拉列表中选择类型或输入自己的类型。在 Name（名称）框中，输入参数的名称。
   - 单击 Add（添加）添加该参数。
   - 对每个方法参数重复该过程。要注意的是，如果要删除已经添加的参数，可以从列表中选择要删除的参数，然后单击 Delete（删除）。要在已有的两个参数之间插入参数，首先选中要出现在新参数后面的参数，当单击 Add 时，则新的参数插入到所选中参数的前面。
   - 当添加完所有的参数之后，单击 OK。
5. 从 Access（访问）下拉列表中，选择一个访问修改符。方法可以声明为 public、protected、private 或 default（软件包）。
6. 选择其他访问修改符，如 abstract、final、static、synchronized 或 native。
7. 要插入用于方法的 Javadoc 注释，在 Javadoc comment（Javadoc 注释）框中输入注释文本。

8. 单击 Add 插入该方法声明到文件中。Class Outline 自动更新用以显示这个新的方法。

### 添加新的成员变量

1. 在 Class Outline 中，右击将定义其成员变量的类名称。在出现的快捷菜单中，单击 Add Member Variable（添加成员变量）。
2. 在 Add Member Variable（添加成员变量）对话框中，在 Method Member Variable（成员变量名）框中输入该成员变量的名称。
3. 从 Data Type（数据类型）下拉列表中选择一种数据类型，或者输入自己的数据类型。
4. 从 Access（访问）下拉列表中，选择一个访问修改符。成员变量可以声明为 public、protected、private 或 default（软件包）。
5. 选择其他的修改符，如 abstract、final、static、synchronized 或 native。
6. 要插入用于成员变量的 Javadoc 注释，在 Javadoc comment（Javadoc 注释）框中输入注释文本。
7. 在 Initial value（初始值）框中输入一个值用来初始化成员变量。
8. 单击 Add 插入该成员变量到文件中。Class Outline 自动更新用以显示这个新的成员变量。

### 添加新类

1. 要添加一个嵌套的类，在 Class Outline 中右击包含的类名称。要添加一个最高层的类，右击在 Class Outline 中的任意类或 Class Outline 窗口本身。

2. 在快捷菜单上，单击 Add Class（添加类）。
3. 在 Add Class 对话框中，在 Class Name 框中输入类名称。
4. 要使得该类成为在 Class Outline 中选中类的嵌套(内部)类，选择 Create a nested class（创建嵌套类）选项。有关嵌套类的信息，请参看在“Java Language Specification”中的“Inner Classes”一部分。
5. 从 Access（访问）下拉列表中，选择一个访问修改符。最高级类可以声明为 public 或 default(软件包)。嵌套类可以声明为 public、protected、private 或 default（软件包）。
6. 选择其他的修改符，如 abstract、final 或 static（只有嵌套类可以声明为 static）。
7. 要插入用于类的 Javadoc 注释，在 Javadoc comment（Javadoc 注释）框中输入注释文本。
8. 单击 Add 插入该类到文件中。Class Outline 自动更新用以显示这个新类。

## 添加接口方法声明

使用 Class Outline，用户可以自动插入用于类所执行接口的方法声明。

### 添加接口方法声明

1. 在 Class Outline，展开 Implemented Interface 节点。
2. 右击接口的名称，并且在快捷菜单中单击 Add Method Subs（添加子方法）。

Class Outline 为在该接口中的每个方法添加一个声明。随后用户可以定义自己的实现。

## 删除定义

使用 Class Outline 可以从 .java 文件中快速删除定义。

### 删除定义

1. 在 Class Outline，右击要删除的项目。
2. 在快捷菜单上，单击 Delete，则该定义从源文件中删除。

> **注意：** 如果使用 Text 编辑器来删除定义，该定义在 Class Outline 中关联的项目被自动删除。

## 移动或复制定义

使用 Class Outline，可以轻松地移动或复制方法、成员变量或类的定义。

### 移动或复制定义

1. 在 Class Outline 中，右击要移动或复制项目的名称。
2. 要移动项目，在快捷菜单上单击 Cut（剪切）；要复制项目，单击 Copy（复制）。
3. 右击指示所要插入定义位置的项目。
   - 要将方法或成员变量定义插入到类结尾，右击该类名称。

- 要将方法或成员变量定义插入类中的另外项目的前面，右击该项目的名字。
- 要插入一个类定义，单击在文件中所要插入定义位置之前的类名称。

4.在快捷菜单上，单击 Paste（粘贴）。

**注意：**如果使用 Text 编辑器来移动或复制定义，则在 Class Outline 中与其相关联的项目自动被移动或复制。

## 超越方法

Class Outline 显示类从超类中继承的所有方法，同样也显示从分层结构中其他类中继承的方法。使用 Class Outline，用户可以快速添加定义来超越一个继承的方法。

**注意：**不能超越标记为静态或结局的方法。

### 超越继承的方法

1.在 Class Outline 中，展开类的 Inherited Members 节点。该节点显示方法和成员变量，但是，只有方法能够被超越。
2.右击想要超越的方法。在快捷菜单中，单击 Override method（超越方法）。
3.该方法的声明插入到该.java 文件中，在这里可以添加自己的实现。

## 设置断点

在集成的调试器中，可以使用 Class Outline 快速设置方法的断点。

**设置断点**

1. 在 Class Outline 中，右击想要在其中设置断点的方法。
2. 在快捷菜单中，选择 Insert Breakpoint（插入断点）。在 Class Outline 中双击该方法，可以在源文件中打开该方法，在其中检查断点的设置。可以看到断点字符出现在边界位置上。
3. 右击该方法并在快捷菜单上选择 Remove Breakpoint（删除断点）可以清除在方法中设置的断点。

有关断点和调试的更多信息，参见第 6 章。

## 使用 WFC Component Builder修改组件

WFC Component Builder 是当用户添加属性和事件到基于 WFC 组件时提供帮助的工具。该构造器将必须的成员变量和方法添加到代码中，并且修改组件的 ClassInfo 类。
使用 WFC Component Builder 用来：

- 添加和删除属性。
- 添加和删除事件。

有关 WFC 的更多信息，请参看第 11 章和第 12 章。有关 Visual J++包括的向导和建立程序的更多信息，请参看第 5 章。

## 添加和删除属性

使用 WFC Component Builder，用户可以轻松地在基于 WFC 的组件中添加和删除属性定义。属性与控制当前属性值的私有成员变量相关联。随后，组件提供公共的 get<PropertyName>和 set<PropertyName>方法来检索并设置该成员变量的值（只读属性与 set<PropertyName>方法没有关联），有关 WFC 属性的更多信息，见本书的第 12 章。

当使用 WFC Component Builder 添加属性时，关联的成员变量和方法添加到类中。该构造器还将属性信息添加到组件的 ClassInfo 类中。ClassInfo 允许关于组件的信息在属性浏览器中公布，如 Properties 窗口。

> **注意：**如果组件没有包含 ClassInfo 定义，WFC Component Builder 将在类中插入一个。

当使用 WFC Component Builder 删除属性时，关联的 ClassInfo 项、成员变量和方法自动从类中删除。

### 打开 WFC Component Builder

1. 在 Text 编辑器中打开组件的源文件。
2. 在 Class Outline 中，右击类名称，然后在快捷菜单中选择 WFC Component Builder。

**添加属性**

1. 在 WFC Component Builder 的 Properties 窗格中，单击 Add。
2. 在 Property Name（属性名）框中，输入该属性的名称。
3. 在 Data Type（属性类型）下拉列表中选择一种数据类型（如果该类是在类路径上或在 Java Package Manager 中可用的，用户可以输入在列表中没有的类类型）。
4. 在 Category（种类）下拉列表中，可以任意选择一种种类。种类允许在 Properties 窗口中组合有关的属性。
5. 在 Description（描述）框中输入任意的描述文本。属性描述在 Properties 窗口中显示。
6. 选择 Read-only Property（只读属性）将该属性设置为只读（set<PropertyName>方法将不会添加到类中）。
7. 要添加关联的成员变量，选择 declare Member Variable（声明成员变量）。Associated Member Variable（关联成员变量）框显示成员变量的名称。
8. 单击 Add。
9. 重复前面所讲的过程添加另一个属性（当在 WFC Component Builder 中添加或删除多个项目时，周期性地单击 Apply 来确保存储你的改变）。
10. 单击 OK 关闭 WFC Component Builder。

**删除属性**

1. 在 WFC Component Builder 的 Properties 窗格中，选择想要删除的属性。

   **注意：**删除属性同时将删除所有关联的方法、成员变量和 ClassInfo 信息。所有添加到该方法中的代码也将丢失。但是，在 WFC Component Builder 关闭之后，可以在 Text 编辑器中选择多级撤消删除操作。在 Edit 菜单上，对每个所要恢复的已删除项目选择 Undo（撤消）。

2. 单击 Delete（删除）。
3. 重复这些步骤来删除其他的属性（当在 WFC Component Builder 中添加或删除多个项目时，应该每隔一段时间就单击 Apply（应用），来确保所做的改变保存起来）。
4. 单击 OK 关闭 WFC Component Builder。

## 添加和删除事件

使用 WFC Component Builder，用户可以轻松地在基于 WFC 的组件中添加和删除事件定义。因为事件使用委托(delegate)来调用它的事件处理程序，组件提供公共的 addOn <EventName>和 removeOn<EventName>方法来添加和删除该委托。组件还定义一个用来激活该事件的保护 on<EventName>方法。有关更多的 WFC 属性的信息，见本书的第 12 章“WFC 编程概念”。

当使用 WFC Component Builder 添加事件时，委托的关联方法和成员变量自动添加到类中。构造器还将事件信息添加到组件的 ClassInfo 类中。

ClassInfo 允许关于组件的信息在属性浏览器中公布，如 Properties 窗口。

> **注意：**如果组件没有包含 ClassInfo 定义，WFC Component Builder 将在类中插入一个。

当使用 WFC Component Builder 删除事件时，关联的 ClassInfo 项、委托和方法自动从类中删除。

### 打开 WFC Component Builder

1. 在 Text 编辑器中打开组件的源文件。
2. 在 Class Outline 中，右击类名称，然后在快捷菜单中选择 WFC Component Builder。

### 添加事件

1. 在 WFC Component Builder 的 Event（事件）窗格中，单击 Add。
2. 在 Event Name（事件名）框中，输入该事件的名称。
3. 在 Type（类型）下拉列表中选择一种事件类型（如果名为 <EventType> 的类和名为 <EventType> 的处理程序是在类路径上或在 Java Package Manager 中，用户可以输入在列表中没有的类型）。
4. 在 Category（种类）下拉列表中，可以任意选择一种分类。分类允许在 Properties 窗口中组合有关的事件。
5. 在 Description（描述）框中输入任意的描述文本。事件描述在 Properties 窗口中显示。
6. 单击 Add。

7. 要添加其他事件，重复上述过程（当在 WFC Component Builder 中添加或删除多个项目时，应该每隔一段时间就单击 Apply（应用）来确保所做的改变保存起来）。
8. 单击 OK 关闭 WFC Component Builder。

**删除事件**

1. 在 WFC Component Builder 的 Event 窗格中，选择想要删除的事件。

   **注意：** 删除事件同时将删除所有关联的方法、委托和 ClassInfo 信息。所有添加到该方法中的代码也将丢失。但是，在 WFC Component Builder 关闭之后，可以在 Text 编辑器中选择多级撤消删除操作。在 Edit 菜单上，对每个所要恢复的已删除项目选择 Undo（撤消）。

2. 单击 Delete（删除）。
3. 重复这些步骤来删除其他的属性（当在 WFC Component Builder 中添加或删除多个项目时，应该每隔一段时间就单击 Apply（应用）来确保所做的改变保存起来）。
4. 单击 OK 关闭 WFC Component Builder。

## 使用 Object Browser浏览软件包和库

Object Browser 提供一种查看 Java 软件包和 COM 库中内容的便利路径。用户可以快速浏览基于 Java 或 COM 的组件，而不需要将其真正地添加到源代码中，甚至可以在没有这些组件的源代码的情况下这样做。

使用 Object Browser，用户可以：

- 查看包含在软件包和库中的类和成员。
- 选择浏览的软件包和库。
- 过滤类和成员。
- 分组和排序类和成员。

将鼠标指向在 View 菜单上的 Other Windows（其他窗口），并且单击 Object Browse 来打开 Object Browser。

## 查看类和成员

Object Browser 提供两个列表用于查看在软件包和库中的类和成员。primary（主）列表可以显示类或方法之一（或全部显示）。当选中主列表中的类时，它的成员显示在 dependent（从属）列表中；但在主列表中的成员被选中时，所有包含该成员的类显示在从属列表中。

### 设置 Object Browser 视图

1. 要切换是否在主窗口中显示类，可以在 Object Browser 上右击，并且在快捷菜单上选择 Show Classes（显示类）。用户还可以单击在 Object Browser 的命令栏上的 Show Classes 按钮。
2. 要切换是否在主窗口中显示成员，可以在 Object Browser 上右击，并且在快捷菜单上选择 Show Member（显示成员）。用户还可以单击在 Object Browser 的命令栏上的 Show Member 按钮。

为了帮助用户区分类和成员，类以粗体显示。注意，默认情况下，Object Browser 在主列表中按照它们各自的软件包和库分类，这些软件包和库名以粗体显示。要切换这些项目是否按照软件包和库分组，可以右击 Object Browser，并且在快捷菜单上选择 Group by Packages/Libraries（按软件包/库分组）。用户还可以单击在 Object Browser 的命令栏上的 Group by Packages/Libraries 按钮。

## 查看类和成员信息

可以使用 Object Browser 来得到有关在软件包和库中类和成员指定的信息：

- 在 Object Browser 中选中类或成员时，下面的描述窗格显示有关该项目和支持到其他类、软件包和库的超级链接的信息。
- 类节点展开时，显示下列的子节点（可用的）：
  - Implemented Interfaces（实现接口），列出类实现的所有接口。
  - Subclasses（子类），列出展开类的所有类。
  - Superclasses（超类），列出所有在继承层次中的类。
  - 当一个 Java 方法在从属列表中显示时，同时显示它的特征（不显示 COM 方法的特征）。

## 查看定义

如果用于软件包或库的源代码在用户计算机上是有效的，则用户可以从 Object Browser 中定位到类和成员的定义。选择项目并且右击 Object

Browser。在快捷菜单上，单击 View Definition（查看定义）。

### 查看隐含成员

Object Browser 能够显示所有 COM 库的隐含成员。要切换到显示隐含成员，右击 Object Browser，并且在快捷菜单上单击 Show Hidden Members（显示隐含成员）。隐含成员以灰色文本显示。

## 选择软件包和库

默认情况下，Object Browser 显示能够从当前解决方案中引用的软件包和库，在解决方案中包含所有的软件包，软件包是由 Java Package Manager(JPM)安装的，并且所有的软件包都在类路径上。还可以将额外的软件包和库添加到 Object Browser 中。

### 选择浏览的软件包和库

1. 右击 Object Browser，并且在快捷菜单上单击 Select Current Packages/Libraries(选择当前软件包/库)，也可以单击在 Object Browser 的命令栏上的 Select Current Packages/Libraries 按钮。
2. Select Current Packages/Libraries 对话框显示可用于浏览的软件包和库。Solution 节点列出了属于解决方案的 Java 项目。Other Libraries and Packages 节点包含下列的子节点：
   - COM Libraries（COM 库），列出了可用于浏览的所有 COM 组件和类型库。

- Java Installed Packages（Java 安装的软件包），列出了所有在类路径上，并且已经由 JPM 安装的软件包。
- Other Java Packages（其他 Java 软件包），列出了其他可用于浏览，但不在类路径上，也不是由 JPM 安装的其他软件包。

3. 选择想要在 Object Browser 中显示的软件包和库，并且清除所要删除的项目。只有父节点也被选中的软件包和库才在 Object Browser 中显示（默认情况下，自动地选择项目来将字段选择所有的父节点）。用灰色选中符号选中的节点表示选中了其中的一些，但不是全部的子项。

   **提示：** 表示在 Object Browser 中将显示哪个软件包和库的最简单的方法就是，在 Object Browser 中这些项目以粗体显示。

4. 要添加项目到 COM Libraries 和 Other Java Packages 节点，单击 Add。
   - 要添加 COM 组件和库，在 Add New Packages/Libraries（添加新的软件包/库）对话框中，单击 Com Libraries 选项卡。当前已在计算机上注册的所有组件和库将列出（要注册其他组件或库，并且将其添加到列表中的方法是单击 Browse（浏览）找到该文件）。选择所要添加的组件和库，然后单击 OK。
   - 要添加其他的 Java 软件包，在 Add New Packages/Libraries（添加新的软件包/库）对话框中，单击 Other Java Packages 选项卡。单击 Browse 找到包含软件包的文件。当软件包添加之后，单击 OK。

**重点：**将 COM 组件或其他的 Java 软件包添加到 Object Browser 中仅仅是允许用户浏览该组件或软件包。要真正在项目中使用 COM 组件，用户必须导入它。有关如何导入 COM 组件的信息，见本书的第 17 章“编译和导入 COM 对象”中的“导入 COM 对象”一节。
要从项目中使用 Java 软件包，用户必须将其添加到类路径上。有关类路径的信息，见第 1 章中的“使用 JVC 编译应用程序”和“设置类路径”。

5. 单击 OK 应用解决方案。

在 Select Packages/Libraries 对话框中选择用来浏览的项目可以持续存在到关闭 Object Browser 之后。

- 在 Other Libraries and Packages 节点下选中的软件包和库将持续存在于每个用户、每台计算机上。例如，如果在同一台计算机上创建新的解决方案，在 Other Libraries and Packages 下选择的软件包和库的集合将通过 Object Browser 显示。但是，如果其他用户登录到该计算机上，可能开始选择不同的软件包和库的集。
- 在 Solution 节点下选择的项目在关闭该解决方案之后将不会继续存在。下次重新打开该解决方案时，默认选择所有在该解决方案中的 Java 项目。

# 分组和排序类和成员

Object Browser 允许用户按类型来分组或排序类和成员。Object Browser 默认不使用任何分组和排序。

## 分组或排序类和成员

1. 右击 Object Browser，并且在快捷菜单上单击 Grouping and Sorting（分组和排序）。用户也可以单击在命令栏上的 Grouping and Sorting 按钮。
2. 要分组或排序类，需要在 Classes 解决方案下拉列表中选择一个组分类：
   - 如果要通过访问级别来分组类，选择 Group by Class Access（按类访问分组）。公共类被分配到 Public Types(公共类型)分类组中；非公共类被分配到 Package Type（软件包类型）分类组中，因为它们只可以从它们从属的软件包中访问。
   - 选择 Group by Class Type（按类型分组）将按它们的类型分组。
   - 选择〈No Grouping〉（<不分组>）来删除类的分组。
3. 如果类被分组，用户可以改变组分类顺序。从列表中选择一种分类，然后单击向上箭头或向下箭头按钮。
4. 如要分组或排序成员，选择在 Memers 解决方案中的下拉列表中选择组分类。

   - 要按照成员的访问级别分组，选择 Group by Member Access（按成员访问分组）。

- 要按照成员的类型分组，选择 Group By Member Type(按成员类型分组)。
- 要删除成员分组，选择<No Grouping>。

5. 如果成员被分组，用户可以改变组分类顺序。从列表中选择一种分类，然后单击向上箭头或向下箭头按钮。
6. 要在 Object 显示组分类的名称，选择 Group with headers in main list(在主列表中带有标题的组）。

   **注意：**组标题只能用于在 Object Browser 的主列表中。例如，假设用户已经将成员分组，但 Object Browser 在主列表中只查看一个类。当选中类并且它的成员显示在辅助列表中时，这些成员被分组，但不包括实际的组标题。有关在主窗格和辅助窗格中查看类和成员的信息，请参看本章前面的“查看类和方法”一部分。

7. 单击 OK。

# 第 4 章　访问数据

使用设计工具和向导，用户可以很方便地在 Visual J++内访问数据。通过添加 WFC 数据控件到在 Forms Designer 中的窗体上，可以快速配置数据检索和显示的方式。还可以运行 Data Form Wizard 来自动生成数据绑定。

Visual J++通过 ActiveX Data Object（ADO，ActiveX 数据对象）控件来访问数据，这是用于 WFC 应用程序的数据编程模块。ADO 对象的内核包含 Connection（连接）、Command（命令）和 Recordset（记录集）对象，利用它们，可以连接到数据库和检索记录集。ADO 还提供 DataSource 组件，该组件将 Connection，Command 和 Recordset 对象的功能组合到一起。

> **注意：** 在 Forms Designer 中的 Toolbox 只提供 DataSource 控件；Connec-tion，Command 和 Recordset 对象只能用在代码中。有关使用 ADO 对象编程的信息，请参见 Visual Studio 在线文档 Microsoft ActiveX Data Object 中的“ADO Tutoral(VJ++)”部分。

ADO 通过 DataBinder 组件支持简单的数据绑定。该组件从 Recordset 或 DataSource 组件中绑定字段到 WFC 控件的属性中。Visual J++还提供复杂数据绑定控件，如 DataGrid 和 DataNavigator 控件，这些控件直接影

响记录集。有关数据绑定的信息，参见本书的第 18 章。
要在窗体上访问数据，用户可以运行 Data Form Wizard（数据窗体向导），或在 Forms Designer 中执行下列步骤：

- 检索记录集
- 使用 DataBinder 控件或 dataGrid 控件绑定数据
- 浏览记录

有关 ADO 的更多信息，见 Visual Studio 在线文档中 Microsoft ActiveX data Object 的“Getting started with ADO 2.0”。有关基于 COM 的数据访问使用 Visual Studio 可用选项的更多信息，见 Visual Studio 在线文档中的“Choosing the Right data Access Technology”。

## 运行 Data form Wizard

使用 Data Form Wizard（数据窗体向导），用户可以自动生成绑定到数据库中字段上的窗体。Data Form Wizard 通过 ADO 检索数据，并且支持 Microsoft Access .mdb 文件和能够通过 ODBC 访问的数据库。

**启动 Data Form Wizard**

1. 要使用 Data Form Wizard 加入数据绑定窗体到应用程序中，可以在 Project Explorer 中右击将包含该窗体的项目或文件夹节点。
2. 在快捷菜单上，指向 Add，然后单击 Add Form（添加窗体）。
3. 在 Add Item（添加项目）对话框中，选择 Data Form Wizard 图标。

4. 在 Name 框中，为窗体输入一个名称。
5. 单击 Open（打开）。

   **注意：**当使用 Application Wizard（应用程序向导）创建一个新的基于数据的应用程序时，也将启动 Data Form Wizard。

6. 在 Data Form Wizard 的 Introduction（介绍）步骤中，可以在下拉式列表中选择 *profile*（如果想要使用的配置文件没有列出来，单击省略号（...）按钮来打开该配置文件）。有关配置文件的更多信息，见本书的第 5 章。
7. 单击 Next（下一步）指定数据库类型。

## 数据库类型步骤

在 Data Form Wizard 中的 Database Type（数据库类型）步骤中，用户指定数据库的格式。

### 指定数据库格式

1. 为所有的 Microsoft Access .mdb 文件，选择 Access（访问）选项。单击 Next 指定数据库文件。
2. 为选择 ODBC 选项通过 ODBC 访问数据库，如 ISAM 数据库（dBase，FoxPro 或 Paradox）或远程数据源（SQL Server 和 Oracle）。单击 Next 指定 ODBC 连接信息。

注意：尽管可以使用 Microsoft Access ODBC 驱动程序来连接到 Access 数据库，但是选择 Access 选项会提供更好的性能。

## 数据库步骤

如果在 Data Form Wizard 的 Database Type（数据库类型）步骤中选择了 Access，那么 Database 步骤允许指定 .mdb 文件。

### 指定 .mdb 文件

1. 在 Database Name（数据库名）框中，输入 .mdb 文件的完整路径和文件名，或者单击 Browse（浏览）按钮来找到该文件。
2. 单击 Next 指定有关创建窗体的细节。

## 连接信息步骤

在 Data Form Wizard 的 Database Type 步骤中如果选择了 ODBC 格式，Connect Information 步骤允许用户指定 ODBC 信息，如数据源名称 (DSN)、数据库和驱动程序。
下列过程显示了通过 ODBC 连接到数据库的不同方法。

### 使用已经创建的 DSN 连接到数据库

1. 在 DSN 下拉式列表中，选择 ODBC 数据源名称（要创建数据源，使用在 Windows Control Panel（控制面板）中的 ODBC 图标）。
2. 如果用户 ID 和密码与 ODBC 数据源关联，分别在 UID（用户 ID）和

PWD（密码）框中输入这些信息。

**使用常规 DSN 连接到数据库**

1.在 DSN 下拉式列表中，选择常规数据源文件的名称，如 FoxPro Files（FoxPro 文件）。
2.在 Database 框中，输入数据库文件的完整路径和文件名。

**使用指定的驱动程序连接到数据库**

1.在 Database 框中，输入数据库文件的完整路径和文件名。
2.在 Driver 下拉式列表中，选择所要使用的驱动程序，如 Microsoft FoxPro Driver（*.DBF）。

**连接到完全服务器数据库**

1.在 UID 和 PWD 框中，分别输入关联的用户 ID 和密码。
2.在 Database 框中，输入数据库文件名。
3.在 Driver（驱动程序）下拉式列表中，选择所要使用的驱动程序，如 Sql Server 或 Oracle。
4.在 Server 框中，输入服务器名称。

一旦指定连接信息之后，单击 Next 来指定要创建窗体的细节。

## 窗体步骤

在 Data Form Wizard 中的 Form（窗体）设置步骤中，用户指定数据绑定

窗体的名称和窗体的设计。

**指定窗体细节**

1. 在 Form name 框中编辑文本来改变窗体的名称。
2. 在 Form layout（窗体设计）下面，选择下列选项之一：
   - Single record（单一记录，默认）。窗体一次显示一条记录。在数据库中的每个非逻辑字段绑定到一个 Edit 控件上；逻辑字段绑定到 CheckBox 控件。Label 控件与绑定控件相关联，并且显示字段的名称。
   - Grid（Datasheet）（网格（数据表））。窗体在 DataGrid 控件中显示多个记录。
   - Master/Detail（主/细节）。窗体显示来自与两个相关联的表（或查询）的数据，所代表的就是一对多关联。窗体在 DataGrid 控件中一次显示一个来自 master（主）表中的记录，及来自 detail（细节）表中的多个记录。
3. 在 Database Connection（数据库连接）下面，选择下列的选项之一：
   - Controls（控件，默认）：窗体使用一个 ADO DataSource 控件来访问数据库。DataSource 控件与 Forms Designer 兼容；如果当向导创建该窗体之后想要改变这些控件的属性，则可以在 Forms Designer 中完成这些修改操作（DataSource 控件是非可视控件，只有当窗体在设计视图中时，它才能够显示，而当运行时，该控件不显示。
   - Code（代码）。窗体使用 ADO Connection 和 Recordset 组件来访问数据库。Forms Designer 不支持这些组件，它们只能用在代码中。

4.单击 Next 指定绑定控件的记录源。

## 记录源步骤

在 Data Form Wizard 的 Record Source（记录源）步骤中，用户选择想要绑定到在窗体上控件的字段。

> **注意：**如果在窗体步骤中选择 Master/Detail（主/细节）设计，Data Form Wizard 提供 Master Record Source（主记录源）和 detail Record Source（细节记录源）。首先使用下列过程来设置 Master record Source 步骤，然后单击 Next 并且为 Detail Record Source 步骤重复这些过程。

### 指定绑定字段

1.在 Record source（记录源）下拉式列表中选择包含所要绑定字段的表名称（当在 Master/Detail 窗体设计中定义主记录源时，选择由公共字段唯一标识的记录。而对于细节记录源，则选择关联表）。
2.Available fields（有效字段）列表包含在指定表中的字段。Selected fields（选中的字段）列表包含将按照列出的顺序绑定到窗体的字段。使用下面列出的按钮在两个列表中移动字段。

| 按钮 | 说明 |
| --- | --- |
| 〉 | 将在 Available Fields 列表中选中的字段移动到 Selected |

Fields 列表中。选中的字段将绑定到在窗体上的控件

>> 将在 Available Fields 列表中的所有字段移动到 Selected Fields 列表中。所有的字段将绑定到在窗体上的控件

< 将在 Selected Fields 列表中选中的字段移动到 Available Fields 列表中。选中的字段将不会绑定到在窗体上的控件

<< 将在 Selected Fields 列表中的所有字段移动到 Available Fields 列表中。没有字段将会绑定到在窗体上的控件

**注意：** Data Form Wizard 向导不能绑定具有二元数据类型的字段。

3. 要改变绑定字段的顺序，在 Selected Fields 列表中选择该字段，并且单击向上或向下箭头。
4. 要排序将要由窗体显示的数据，在 Column to sort by（按列排序）下拉式列表中选择该字段。
5. 如果窗体风格是 Single record（单一记录）或 Grid(Datasheet)（网格（数据表）），单击 Next 来加入附加的控件到窗体。
   或者
   如果窗体风格是 Master/Detail，并且已经完成了选择主/细节记录源的操作，单击 Next 指定主/细节的关系。

## 记录源关系步骤

如果在 Data Form Wizard 的窗体步骤中选择了 Master/Detail 风格，Record Source Relation(记录源关系)步骤允许用户在主/细节记录源中指定一对

多关系。

**指定主/细节关系**

1. 在 Master（主）列表中选择相关的字段。该字段应该唯一标识在主记录源中的每个记录。
2. 在 Detail（细节）列表中选择相关的字段。
3. 单击 Next 来加入另外的控件到窗体。

## 控件选择步骤

在 Data Form Wizard 中的 Control Selection（控件选择）步骤中，用户可以选择想要出现在窗体上的其他控件。这些控件所需要的代码自动加入到窗体。

**加入附加的窗体**

1. 在 Available controls（可用控件）下，选择所要加入的控件：

| 控件 | 说明 |
|---|---|
| Add（添加）按钮 | 允许用户向数据库中加入新的记录 |
| Delete（删除）按钮 | 允许用户从数据库中删除记录 |
| Update（更新）按钮 | 使用在窗体中所做的改变更新数据库 |
| Refresh（刷新）按钮 | 使用数据库中的最后的记录刷新窗体 |
| Close（关闭）按钮 | 允许用户关闭窗体 |
| Data（数据）定位 | 允许用户在记录中定位 |

2. 单击 Next 来查看窗体的摘要信息。

## 摘要步骤

在 Data Form Wizard 的 Summary（摘要）步骤中，用户可以存储和复查向导的设置。

**存储和复查向导设置**

1. 要存储设置到已有的配置文件中，在下拉式列表中选择该配置文件。要存储设置到一个新的配置文件，单击省略号按钮（...）来指定文件名称（有关配置文件的更多信息，见本书的第 5 章）。
2. 要复查设置，单击 View Report（查看报告）。要存储该报告，单击在 View Report 对话框中的 Save 按钮。
3. 单击 Finish（完成）按钮将该窗体加入到项目中。

Data Form Wizard 提供快速开始的方法来创建带有绑定到数据库中字段控件的窗体。也可以不使用向导，而直接使用 DataBinder 控件或 DataGrid 控件来绑定数据。

## 检索记录集

Visual J++提供 DataSource 控件在 Forms Designer 中访问数据。该控件允许用户连接并查询数据库来检索记录集。

**注意：**要在代码中检索记录集，还可以使用 Connection，Command 和 Recordset 组件。Forms Designer 还支持单独的 DataSource 组件。有关使用 ADO 组件编程的信息，参见在 Microsoft ActiveX Data Objects 中 Visual Studio 在线文档的“ADO tutorial(VJ++)”。

### 使用 DataSource 控件检索数据

1. 在 Project Explorer 中，双击窗体名，在 Forms Designer 中打开该窗体。
2. 在 Toolbox 中，单击 WFC Controls 选项卡。在 Toolbox 中单击 DataSource 控件，然后单击窗体来加入该控件。

   **注意：**因为 DataSource 控件只检索数据，但不显示它们，所以在运行窗体时该控件不可见。

3. 要连接到数据库，设置 DataSource 控件的 connectionString 属性。
   1. 在窗体上选择 DataSource 控件。
   2. 在 Properties 窗口中，单击 connectionString 属性，然后单击省略号（...）按钮来打开 Data Link Properties 对话框。
4. 要访问名为 ODBC 的数据源：
   - 单击 Provider（提供者）选项卡，并且选择 Microsoft OLE DB Provider for ODBC Drivers。
   - 单击 Connection（连接）选项卡。在步骤 1 中，选择 Use data Source name（使用数据源名），并且在下拉式列表中选择数据源名称。如果需要的话，在步骤 2 中，可以输入用户名和密码。

或者

不使用 ODBC 而直接访问 Microsoft Access 的.mdb 文件：

- 单击 Provider 选项卡，并且选择 Microsoft Jet 3.51 OLE DB Provider。
- 单击 Connection 选项卡，在步骤 1 中，输入数据库的完整路径名和文件名，或者单击省略号（...）按钮来浏览该文件。在步骤 2 中，在适当的位置可以输入用户名和密码。

5. 单击 OK 建立数据库连接。
6. 要查询该数据库，设置 DataSource 控件的 commandText 属性为一个 SQL 串。例如，要从表中检索所有命名为 Products 的记录，输入 Select * from Products。

现在用户已经检索出了一个记录集，必须使用 DataBinder 控件或 DataGrid 控件绑定它。

## 使用 DataBinder控件绑定数据

DataBinder 控件从记录集中绑定字段到其他控件的属性中。当属性绑定时，它将自动以当前记录中该字段的值设置。下列的过程显示如何绑定一个 Edit 控件的文本属性。

### 使用记录集关系 DataBinder 控件

1. 添加 DataSource 控件到窗体中用来检索数据。有关如何操作的信息，见前面的一节“检索记录集”。

2.加入 Edit 控件和 DataBinder 控件到窗体上。

**注意：**与 DataSource 控件一样，DataBinder 控件在窗体运行时也不显示，因为它只管理绑定操作，而不是显示数据。

3.在窗体上选择 DataBinder，并且设置它的 dataSource 属性：

- 在 Properties 窗口中，单击 dataSource 属性。
- 在下拉式列表内，选择 DataSource 控件名。

DataBinder 控件的绑定 bindings 属性标识了当前已经定义的绑定。可以使用控件的属性页或其绑定编辑器设置这些属性。

**使用属性页创建绑定**

1.在窗体上选择 DataBinder 控件。在 Properties 窗口中，单击 Property Page（属性页）工具栏按钮。
2.在属性页的 Data Field（数据字段）下拉式列表中，选择绑定的数据字段名。
3.在 Control(控件)下拉式列表中，选择 Edit 控件名。
4.在 Property（属性）下拉式列表中，选择文本的属性。
5.单击 Add。该文本属性现在绑定到数据字段。
6.要加入其他绑定，重复该过程。当加入所有的绑定之后，单击 OK。

**使用绑定编辑器创建绑定**

1.在窗体上选择 DataBinder 控件。在 Properties 窗口中，单击 bindings(绑定）属性，然后单击省略号（...）按钮。

2. 要在绑定编辑器中加入绑定，单击 Add（对于每个所要添加的绑定，单击 Add）。
3. 单击 OK。
4. 在 Properties 窗口中，展开 Bindings 属性。每个绑定按照它的 index（索引）顺序列出来，索引指出了添加绑定的顺序（第一个绑定的索引为 0）。
5. 要定义该绑定，展开它的索引条目。
6. 单击 fieldName（字段名）。在下拉式列表中，单击绑定的数据字段名。
7. 单击 target（目标）。在下拉式列表中，选择 Edit 控件名。
8. 单击 PropertyName（属性名）。在下拉式列表中，选择文本属性。

一旦创建了绑定，用户就可以添加 DataNavigator 控件到窗体上，用来浏览记录。用户还可以在绑定中应用数据格式用来显示数字、日期或逻辑值。

有关 DataBinder 控件和有关编程的实例，见本书第 18 章中的“DataBinder 组件”一节。

## 使用 DataGrid 控件绑定数据

DataGrid 控件从记录集中绑定字段，并且在连续的行和列中显示这些数据。在设置控件的 dataSource 属性到 DataSource 控件时，该控件自动出现。

在 DataGrid 控件中显示的数据总是与在记录集中的数据同步，反之亦

然。记录集的 cursorType 和 lockType 属性确定数据是否动态反映数据库中的数据，及在记录集中的数据是否能够改变。

**绑定 DataGrid 控件到记录集**

1. 添加 DataSource 控件到窗体中用来检索数据。有关如何操作的信息，见前面的“检索记录集”一节。
2. 添加 DataGrid 控件到窗体上。
3. 设置 DataGrid 控件的 DataSource 属性：
   - 在属性窗口中，单击 dataSource 属性。
   - 在下拉式列表中，选择 DataSource 控件名。

注意，DataGrid 控件在设计视图中显示活动数据。记录集中的当前记录由网格对应行标题中的标记来标识。在窗体中添加 DataNavigator 控件，可以快速在网格中浏览。

有关在代码中使用 DataGrid 控件的信息，见在 Microsoft Visual J++ 6.0 Reference Library 中第一部分 Microsoft Visual J++ 6.0 WFC Library Reference 中的 com.ms.wfc.data.ui 软件包。

## 访问列属性

通过在 Properties 窗口中设置 DataGrid 控件的属性，用户可以快速配置该控件的外观和功能。在网格中的每列也有自己的属性设置，用户可以在网格属性中访问它们。

### 访问列的属性

1. 在 Properties 窗口中，展开 DataGrid 控件的 columns（列）属性。每一列按它在网格中的索引列出，索引指出显示列的顺序（第一列的索引为 0）。
2. 要显示列属性，展开它的索引条目。

默认情况下，DataGrid 控件为基本记录集中的每一字段包含一列，并且列按照记录集中字段的顺序列出。但是，可以在 DataGrid 控件已经创建之后很简单地添加、删除和重新安排列。还可以将数据格式应用到列中，用来显示数字、日期或逻辑值。

## 添加、删除和重新安排列

DataGrid 控件提供了列编辑器，用来在网格中添加、删除和重新安排列。默认情况下，基本记录集中的每个字段都绑定到网格中的某一列上，并且列按照字段在记录集中的顺序排列。

### 在网格中添加、删除和重新安排列

1. 在 Properties 窗口中单击 DataGrid 控件的 columns 属性，打开列编辑器。然后单击省略号（…）按钮。
2. 要添加列，单击 Add。

**注意：**所添加的列在最初是非绑定的。要绑定该列，则需要将它的boundFieldName属性设置为记录集中的一个字段名（有关如何访问列属性的信息，见前面的“访问列属性”一节）。

3. 在列表中选择列，并且单击Remove（删除）来删除列。

   **注意：**如果只是想临时隐藏某列，而不是将其从网格中完全删除，则可以设置它的可见的属性为假。

4. 选择所要移动的列，然后单击Up（向上）和Down（向下）来重新安排列。
5. 当完成了添加、删除和重新安排列的操作时，单击OK。

## 格式化数据

当使用DataBinder控件或是DataGrid控件绑定数据时，用户可以格式化数字、日期和逻辑值的显示。每个由DataBinder控件绑定的数据及每个在DataGrid控件中的列都具有dataFormat属性，该属性允许用户指定格式。

### 访问dataFormat属性

1. 如果正在使用DataBinder控件，在Properties窗口中展开它的bindings（绑定）属性。
   或者

如果正在使用 DataGrid 控件，在 Properties 窗口中展开它的 columns 属性。

2. 每个绑定或列按照它的索引顺序列出，索引指出了它在父控件中的顺序（第一个绑定或列的索引为 0）。
3. 展开绑定或列中所要格式化的索引条目，然后单击 dataFormat。

### 格式化数字值

1. 设置 dataFormat 属性为 NumberDataFormat。
2. 展开 dataFormat，并且设置 format（格式化）属性为一个数字格式化串。有关可用的格式化串的信息，见在 Microsoft Visual J++ 6.0 Reference Library 中第一部分 Microsoft Visual J++ 6.0 WFC Library Reference 中的 com.ms.wfc.data 软件包。

### 格式化日期（或时间）值

1. 设置 dataFormat 属性到 DateDataFormat。
2. 展开 dataFormat。单击 Format 属性，然后在下拉式列表中选择下面所列出的值中的一种：

| 格式 | 说明 |
| --- | --- |
| Long | 使用在 Windows 控制面板中区域设置中的 Long Date（长日期）设置格式化日期（默认的长日期设置显示为 Monday , March 9 , 1998） |
| Short | （默认）使用在 Windows 控制面板中区域设置中的 Short Date（短日期）设置格式化日期（默认的短日期设置显 |

| | |
|---|---|
| | 示为 3/9/98) |
| Time | 使用在Windows控制面板中区域设置中的Long Time(长时间)设置格式化时间(默认的长时间设置显示为 2:45:05 PM) |
| Custom（定制） | 使用定制格式来格式化日期或时间。将 customFormat 属性设置为定制的串。有关可用的格式化串的信息，见在 Microsoft Visual J++ 6.0 Reference Library 中第一部分 Microsoft Visual J++ 6.0 WFC Library Reference 中的 com.ms.wfc.data.ui 软件包中的 DataDataFormat.setCustomFormat |

**注意：** customFormat 属性的设置只能应用在格式化属性设置为 Custom 时。如果格式化属性设置为 Short，Long 或 Time，所有用于 customFormat 的设置将被忽略。

## 格式化逻辑值

1. 设置 dataFormat 属性到 BooleanDataFormat。
2. 展开 dataFormat，并且设置下列的属性：
   - falseValue，指定当值为假时显示的字符串。默认的字符串为 False。
   - nullValue，指定当值为空时显示的字符串。默认的字符串为（空）。
   - trueValue，指定当值为真时显示的字符串。默认的字符串为 True。

## 定位记录

DataNavigator 控件允许用户改变在记录集中的当前记录。这要将 DataNavigator 控件与其他数据绑定控件，如 DataBinder 控件，一起使用。DataBinder 控件绑定其他控件的属性到记录集中的字段上。该属性从记录集的当前记录（最初是开头的记录）中获得数据。使用绑定到同一个记录集上的 DataNavigator 控件来移动到其他的记录上。

### 使用 DataNavigator 控件定位记录集

1. 添加 DataSource 控件到窗体中用来检索数据。有关如何操作的信息，见前面的“检索记录集”一节。
2. 使用 DataBinder 控件和一个 Edit 控件从与 DataSource 控件关系的记录集中绑定数据。有关如何操作的信息，见本章前面的“使用 DataBinder 控件绑定数据”一节。
3. 将 DataNavigator 控件添加到窗体上。
4. 将 DataNavigator 控件的 dataSource 属性设置为 DataSource 控件名。
5. 使用 DataNavigator 控件中的按钮来定位数据：

| 按钮 | 定位指示 |
|---|---|
| << | 移动到第一个记录 |
| < | 移动到前一个记录 |
| > | 移动到下一个记录 |
| >> | 移动到最后的记录 |

有关在代码中使用 DataNavigator 的信息，见在 Microsoft Visual J++ 6.0 Reference Library 中第一部分 Microsoft Visual J++ 6.0 WFC Library Reference 中的 com.ms.wfc.data.ui 软件包中“DataNavigator Control”。

# 第 5 章　向导和生成器简介

Visual J++提供几种向导和生成器来帮助用户开发应用程序。向导指导用户通过一系列的步骤，从而添加新的文件到项目中。每一步骤都包含Back（上一步）和 Next（下一步）按钮，用来在这些步骤中前后移动。Cancel（取消）按钮取消所做的所有设置并且关闭向导。Finish（完成）按钮接受当前输入的选项，并且对剩余的步骤使用默认的设置（Finish按钮只有在用户对向导作了足以完成任务的选择之后才有效）。

> **注意：**当运行向导时，用户可以将设置存储到一个配置文件中。下次使用向导时可以装入这些配置文件来重新使用这些设置。

当向导通过一步一步的过程创建了新的文件之后，生成器帮助用户修改在项目中已有的文件。Visual J++提供了下列的向导和生成器。

| 向导或生成器 | 说明 |
|---|---|
| Application Wizard（应用程序向导） | 自动创建包含窗体的 WFC 应用程序。其中有将窗体绑定到数据库中字段的选项。有关更多的信息，见第 1 章中的“使用 Application Wizard 创建 Windows 应用程序”一节 |
| WFC Component Builder（WFC 组件编码程序） | 通过添加或删除属性或事件来修改 WFC 组件。有关更多的信息，见第 3 章中的“使用 WFC Component |

| | |
|---|---|
| | Builder 修改组件”一节 |
| Data Form Wizard（数据窗体向导） | 自动生成绑定到数据库中字段的窗体。Data Form Wizard 支持 Microsoft Access .mdb 文件和能够通过 ODBC 访问的数据库。有关更多的信息，见第 4 章中的“运行 Data Form Wizard”一节 |
| J/Direct Call Builder（J/直接调用编码程序） | 安照适当的@dll.improt 标记插入用于 Win32 API 函数的 Java 定义到代码中。有关更多的信息，见第 19 章中的“J/Direct Call Builder”一节 |

# 第 6 章　调试应用程序

在代码中出现语法错误时，Visual J++编译器通过显示指出应用程序没有成功更新的消息，并且在 Visual J++任务列表中显示错误列表来警告用户。

但是，通常发生在代码中的错误并不是语法错误，而是一些如死循环这样的逻辑错误。要处理这样的错误，用户可以使用 Visual J++调试器所支持的特性。

在本章后面的“基本调试过程”一节中，用户可以学到使用与 Visual J++项目集成的调试器的知识。这些过程显示了如何设置断点、代码步骤、查看成员变量的运行值等。

在线文档也提供了已创建的方案，用来显示如何使用调试器用于 Visual J++项目的指定类型的属性。在下列方案中，用户能够发现整个过程中重新创建项目的细节。

- 调试 WFC Application（WFC 应用程序）
- 调试 Console Application（控制台应用程序）
- 调试 Multithreaded Applications（多线程应用程序）
- 调试 Java COM Object（Java COM 对象）
- 调试 Java Applet（Java 小应用程序）

## 调试过程

Visual J++调试支持包括断点、中断表达式、监视表达式、线程控制和异常处理。用户还可以一次单步通过代码的一条语句或方法，并观察变量和属性值。

调试没有诀窍，也没有适合于各种情况的固定步骤。基本上，调试可以帮助用户理解当应用程序运行时所发生的事情。调试工具提供了应用程序当前的整体状态，这些状态包括：

- 应用程序的用户界面的外观
- 成员变量、表达式和属性的值
- 有效方法调用

## 基本调试过程

已经调试了几个应用程序之后，可能会发现几乎所有的程序都要执行某些过程。

## 输入命令行参数

当使用需要启动参数来调试的可执行文件时，用户可以在命令行，或开发环境中输入这些参数。

下面列出的过程解释了如何使用在项目的 Properties 对话框中环境的

Launch（开始）选项卡上 Custom（定制）选项。可以从 Project 菜单中访问打开项目对应的 Properties 对话框。

**输入命令行参数**

1. 从 Project Explorer 中，双击 .java 文件来将其装入到 Text 编辑器中。
2. 在 Project 菜单上，单击 <*Project name*>properties（〈项目名〉属性）来显示 Properties 对话框。
3. 在 Launch 选项卡上，单击 Custom 单选按钮。
   在该选项卡上的 Custom 选项提供一个输入用于应用程序的命令行参数的位置。
4. 在 Arguments（参数）文本框中，输入所要传递到程序的信息。
   例如，如果程序以 MM DD YY 格式来接收日期，则对 July 10, 1997 应输入为 7 10 1997。

   **注意：**Arguments 文本框可能显示一些查看器选项。如果有理由改变这些预先设定的选项，现在就可以这样做，然后，在改变的选项后输入日期。

现在，已经做好调试应用程序的准备了。在开始这些过程之前设置一个断点是启动启动此过程唯一途径。

## 显示调试工具栏

用户可以通过调试工具栏来访问调试器最常用的命令和窗口。

**显示调试工具栏**

- 在 View 菜单上单击 Toolbars（工具栏），然后选择 Debug（调试）。

## 设置断点

断点在代码中指定一个位置，当过程执行到该位置时停止，从而可以检查该过程的代码、变量和（有些情况下）寄存器值。另外，用户可以根据需要改变代码，然后继续或终止过程的执行。

### 在启动调试器之前设置断点

当创建并建立了 Visual J++项目之后，用户可以在代码中设置断点。断点在代码中指定代码临时停止执行的位置，以便逐行单步通过代码来检查参数变量值，检查应用程序的动作。可以在应用程序中发生错误的位置设置一个断点，从而在应用程序发生错误之前就暂停程序的执行。

**设置断点并启动调试器**

1. 在编辑器中打开源代码文件，将光标放在应用程序、小应用程序或组件中任何可执行的代码上。
2. 右击鼠标按钮，然后从快捷菜单中选择 Insert Breakpoint（插入断点）。
   一个实心的红色圆形标记放置在编辑器的页边上，指出断点设置在刚刚选中的语句上。
3. 在 Debug 菜单上，单击 Start（启动）或使用键盘快捷键 F5。
   程序开始运行，并且在执行设置断点行的前一行停止（中断）。在页

边上的黄色箭头指出下一个语句准备执行

## 当程序在中断模式中时，用户可以：

- 设置其他断点。
- 检查一个和多个调试窗口来查看程序成员变量的值。

### 重新开始调试工作

- 在 Debug 菜单上单击 Continue(继续)。
  或者
  按下键盘快捷键 F5 运行到下一个断点。
  或
  按下 F11 键按步执行到下一个语句。

### 结束调试工作

- 在 Debug 菜单上单击 End（结束 ）。
  有关如何结束调试工作的更多信息，见 Visual Studio 在线文档中的“Ending a Debugging Session”。

## 在 Breakpoints（断点）对话框中设置断点

学到这里，用户已经可以通过选择在快捷菜单中的选项来添加并删除断点。用户还能在同样的菜单中禁用或启用断点，以及改变断点的属性。这个集成的调试器还提供了一个 Breakpoints 对话框，在该对话框中给出了更多的控件用来管理应用程序的断点。

# 使用调试窗口检查信息

下面讨论单独的调试窗口和每个窗口包含的用途。在这里，用户可以还可以找到用于访问和使用每个调试窗口的特定的过程。

## 在 Auto 窗口中查看信息

Auto 窗口显示在当前执行的方法作用域中的所有变量值。在这里，Locals（局部）窗口中显示用于单线程的变量，Auto 窗口显示所有线程的变量。用这些窗口可以了解可能由于代码在不同的线程上执行所导致的变量的变化。变量在窗口中保持可见的时间同它在作用域中所保持的时间一样长，从而反映该变量的所有变化。当变量移出作用域时，在窗口中的该变量也被删除。

Auto 窗口只有在挂起执行时才被更新。例如，在用户添加一个新的观察变量时、在改变变量的值时或是从快捷菜单中在 10 进制和 16 进制直接切换时，Auto 窗口才更新。在最后一次挂起之后改变的值以红色显示。

### 显示 Auto 窗口

- 在 View 菜单上，单击 Debug Windows（调试窗口），然后选择 Auto（自动）。
  或者
  按下组合键 CTRL+ALT+A
  或者
  单击在 Debug 工具栏上的 Auto 按钮。

**将变量从 Auto 窗口中复制到其他窗口**

- 拖动选中的变量到 Immediate（当前）窗口或是 Watch（观察）窗口。

**改变变量值**

- 单击该值并且编辑它。

**查看或隐藏类成员变量和数组元素**

- 展开或折叠类和数组成员变量来查看或隐藏这些元素。
  见本章后面的“为 Auto，Locals 和 Watch 窗口定位键盘快捷键”一节，在该节中介绍了如何在这些窗口之间中移动，及如何展开和折叠类成员变量和数组元素。

有关 Auto 调试窗口的更多信息，见 Visual Studio 在线文档中的“Viewing Variables in Auto Windows”。

## 在 Locals（局部）窗口中查看信息

Locals 窗口显示在当前堆栈框架中每个方法的类成员变量和它们的值。当执行过程从一个方法切换到另一个方法时，Locals 窗口的内容改变，以反映当前方法的局部变量。

Locals 窗口包含每个方法变量的 Name（名称）、Value（值）和 Type（类型）。在所给出的变量已经在程序中声明之后，该变量的值将显示。如果显示在 Locals 窗口中的变量不在作用域中时，窗口中变量名字后面的 Value 字段中出现一个消息。在看到这些变量值之前，用户需要单步执行更多的代码。只有在挂起执行时，Locals 窗口才更新。在最后一次挂

起之后改变的值以红色显示。

**显示 Locals 窗口**

- 在 View 菜单上，单击 Debug Windows，然后选择 Locals。
  或者
  按下组合键 CTRL+ALT+L
  或者
  单击在 Debug 工具栏上的 Locals 按钮。

**改变在 Locals 窗口中的类信息**

- 从在 Locals 窗口顶部的下拉式列表中选择其他的类。

  **注意：**当在程序中定义了多个类时，用户也许想要查看变量在所有这些类中的值。Local 窗口总是显示用于包含程序的入口点（main()或 init()）的类的成员变量。一旦停止单步进入或停止在不同的类中的断点上，使用在 Locals 窗口中顶部的下拉式列表来选择这些类，并且查看它的成员变量。

**从 Locals 窗口中复制变量到其他的窗口**

- 拖动选中的变量到 Immediate 窗口或 Watch 窗口。

**改变变量值**

- 单击该值并且编辑它。

**查看或隐藏类成员变量和数组元素**

- 展开或折叠类和数组成员变量来查看或隐藏这些元素。
  见本章后面的“为 Auto，Locals 和 Watch 窗口定位键盘快捷键”一节，在该节中介绍了如何在这些窗口之间移动，及如何展开和折叠类成员变量和数组元素。

有关 Local 调试窗口的更多信息，见 Visual Studio 在线文档中的“Viewing Local Variables in Locals Windows”。

## 在 Watch 窗口中查看信息

在 Watch 窗口中，可监视在运行时间中变量、属性和表达式的值。许多调试问题不会立即追踪到某个单独的语句上，所以，用户需要使用监视表达式来观察变量或表达式在整个过程中的行为。

### 显示 Watch 窗口

- 在 View 菜单上，单击 Debug windows，然后选择 Watch。
  或者
  按下组合键 CTRL+ALT+W
  或者
  单击在 Debug 工具栏上的 Watch 按钮。

Watch 窗口跟踪应用程序中在不同时间或位置可能改变的成员变量。

### 在设计时或在中断模式时添加监视表达式

- 从 Text 编辑器、Locals 窗口或 Immediate 窗口中拖动表达式或变量到

Watch 窗口。

**改变变量值**

- 单击该变量来编辑它的值。

**查看或隐藏类成员变量和数组元素**

- 展开类或数组成员变量来查看这些元素。
- 折叠类和数组成员变量来隐藏这些元素。

  见本章后面的“为 Auto，Locals 和 Watch 窗口定位键盘快捷键”一节，在该节中介绍了如何在这些窗口之间中移动，及如何展开和折叠类成员变量和数组元素。

有关 Watch 调试窗口的更多信息，见 Visual Studio 在线文档中的“Inspecting variables and Properties with Watch Windows”。

浏览 Auto，Locals 和 Watch 窗口的键盘快捷键

用户可以使用下面列出的键来查看或隐藏选中类或数组的成员变量，或者在 Auto，Locals 或 Watch 窗口中到处移动。

| 移动到 | 使用 |
|---|---|
| 折叠成员变量列表 | 左箭头键 |
| 查看成员变量 | 右箭头键 |
| 在隐藏变量和查看变量之间切换 | 回车键 |
| 在成员变量列表中向上移动 | 向上箭头键 |
| 在成员变量列表中向下移动 | 向下箭头键 |

| | |
|---|---|
| 在展开的列表上移 | 左箭头键 |
| 在展开的列表下移 | 右箭头键 |

**注意：**可以通过拖动列标题的边界来改变列的大小，将其向右拖动，则使列增大；向左拖动，则使列变小。

## 在 Immediate（立即查看）窗口中查看信息

可以使用 Immediate 窗口来查看和改变值。在该窗口中，可求任何一个表达式、变量或对象的值，并且看到返回的值，或者看到输入命令的效果（必须在当前执行的代码语言中）。
一旦在中断模式中，用户可以将焦点移动到 Immediate 窗口来检查数据。可以在 Immediate 窗口中求任何有效的表达式的值，包括影响属性或变量的表达式。当前有效的窗体或类决定作用域。

### 显示 Immediate 窗口

- 在 View 菜单上，单击 Debug Windows，然后选择 Immediate。
  或者
  按下组合键 CTRL+ALT+I。
  或者
  单击在 Debug 工具栏上的 Immediate。

用户可以将表达式输入到 Immediate 窗口中，用于显示或改变变量、属性和表达式的值。

### 添加表达式或变量到 Immediate 窗口

- 从 Text 编辑器或从 Locals 或 Watch 窗口中拖动变量或表达式到 Immediate 窗口中。

**在 Immediate 窗口中求表达式或变量的值**

1. 在 Immediate 窗口中，将插入点移动到想要求值的变量或表达式上。

   **注意：**要检查包含脚本代码的变量或表达式，在变量或表达式显示它的值之前插入一个问号“？”。

2. 键入 Enter。

   如果选择赋值变量，Immediate 窗口将返回当前值；如果选择表达式，则返回赋值结果。

**再次执行某个语句**

1.将插入点移回到想要再次执行的语句上。
2.如果需要的话，可以编辑当前的语句来改变它的效果。
3.按下 Enter。

**在 Immediate 窗口中移动**

- 使用鼠标或方向键。

  **提示：**除非插入点正在想要执行的语句上，否则不要按 Enter 键。

有关 Immediate 调试窗口的更多信息，见 Visual Studio 在线文档中的“Executing Commands and Evaluation Expressions in the Immediate

Window”。

## 在 Threads 窗口中查看信息

对于多线程应用程序，Threads 窗口允许用户改变执行的线程，或者在任何附加的过程中查看线程。其他调试窗口内容会改变，以反应选中线程的效果。有关更多的信息，见 Visual Studio 在线文档中的“Threads Window”和“Controlling Threads”。

### 显示 Threads 调试窗口

- 在 View 菜单上，单击 Debug Windows，然后选择 Threads。
  或者
  按下组合键 CTRL+ALT+H.
  或者
  单击在 Debug 工具栏上的 Threads。

### 在 Threads 调试窗口中观察线程活动

用于该过程的应用程序示例在 main()方法中创建两个线程，并且将输出显示到 JVIEW 控制台窗口。这些过程涉及到在“多线程 Beverages 应用程序”中创建的代码。如果想要创建这些项目来完成下面列出的过程，现在就做。用户还可以使用下列过程来观察任何多线程应用程序的线程活动。

### 在 Threads 调试窗口中观察线程活动

1. 在 Coffee 类中的下面语句上设置断点：

```
System.out.println(" I Like Coffee" + "   " + i);
```

2. 在 Tea 类中的下面语句上设置断点：

```
System.out.println(" I Like Tea " + "   " + I);
```

3. 在 Class1 的 main()方法中，将插入点放到下面语句上：

```
Coffee m_Coffee = new Coffee(); //creates Coffee object
```

4. 单击鼠标右键，然后在快捷菜单中选择 Run To Cursor（运行到光标处）。
5. 当应用程序进入到断点模式时，打开 Threads 调试窗口。
   注意 main()线程是当前在该窗口中运行的唯一线程。
6. 按下 F5 键来运行到设置在 Coffee 类中的断点。

在 Threads 调试窗口中出现的更多信息以识别应用程序的三个线程 main，Thread-0，Thread-1 的名称、代码中的位置和挂起状态。黄色的箭头标识正在运行的线程，在此时它是 Coffee.run()的 Thread-0。注意 main()方法的线程不再运行，因为它已经传递控件到 Coffee 和 Tea 类的线程中。

> **注意**：尽管这些方案建议 Coffee 线程将成为运行的第一个线程，但并不总是这种情况。操作系统将最终决定在给出的应用程序中线程的过程顺序，除非在代码中的语句指定创建及运行新线程的条件。

7. 按下 F11 键执行当前的语句。
8. 切换到 JVIEW 的控制台窗口，于是输出：

I Like Coffee 0

9. 按下 F5 键运行到设置在 Tea 类中的断点。
10. 按下 F11 键执行当前的语句，于是在 JVIEW 的控制台窗口中显示下列输出：

I Like Tea 0

11. 继续交替地按 F5 和 F11 键，直到下面列出的输出语句之一出现在 JVIEW 的控制台窗口中：

I Like Coffee 9

or

I Like Tea 9

注意，当线程被破坏时，在 Threads 调试窗口中出现有关线程不再出现的信息。

在观察应用程序的每个线程的过程中，用户可能观察到某个线程的行为与预期的不同。要通过挂起其他线程而隔离给定的线程，见下一部分“从 Threads 调试窗口中挂起和继续执行线程”。

## 从 Threads 调试窗口中挂起和继续执行线程

用于本过程的应用程序示例在 main()方法中创建两个线程，并且显示输出到 JVIEW 控制台窗口中。这些过程涉及到在“多线程 Beverages 应用程序”中创建的代码。如果想要创建这些过程来完成下面列出的过程，

现在就做。用户还可以使用下列的过程来挂起和继续执行任何多线程应用程序的线程。

**挂起线程**

1. 在执行已经创建的应用程序的线程之后的语句中设置断点。在本过程示例中，断点设置在 Coffee 类中的下列语句上：

   ```
   System.out.println(" I Like Coffee" + "   " + i );
   ```

   和

   在 Tea 类的下列语句上：

   ```
   System.out.println(" I Like Tea " + "   " + i );
   ```

2. 在 Debug 菜单上，单击 Start，或者按下键盘快捷键 F5 来运行应用程序，同时启动调试器。
3. 当应用程序进入到中断模式时，打开 Threads 调试窗口。

   **注意：** 操作系统将最终决定给定应用程序中线程的处理顺序，除非在代码中的语句指定创建及运行新线程的条件。

4. 在 Threads 调试窗口中，突出显示其中的一个线程，例如，突出显示 Tea.run 的 Thread-1。
5. 单击鼠标右键，然后从快捷菜单中选择 Suspend（挂起）。

   注意，Thread-1 的 Suspension Count（挂起计数）由 0 改变为 1。
6. 通过按 F5 或 F11 键，继续运行应用程序，至少要挂起一个线程。

如果正在使用为该过程创建的应用程序，则在 JVIEW 的控制台窗口中将只看到非挂起线程的输出。

在多线程应用程序调试中的某些点上，用户也许需要重新启动被挂起的线程。按下面的步骤继续执行挂起的线程：

**注意：** 这些过程需要用户有：

3. 打开的线程调试窗口。
4. 在该窗口中有挂起的线程。

**继续执行挂起的线程**

1. 在 Threads 调试窗口中，突出显示一个挂起的线程。在本例中，突出显示 Tea.run 的 Thread-1 线程。
2. 右击鼠标按钮，然后从快捷菜单中选择 Resume（继续执行）。
   注意，Thread-1 的 Suspension Count 由 1 变为 0。

## 在 Call Stack（调用堆栈）窗口中查看信息

Call Stack 窗口显示用于当前执行线程的所有活动的过程或堆栈的列表。活动过程是未执行完的程序。

**显示 Call Stack 窗口**

- 在 View 菜单上，单击 Debug Windows，然后选择 Call Stack。
  或者
  按下组合键 CTRL+ALT+C。

或者
单击在 Debug 工具栏上的 Call Stack 按钮。

**从 Call Stack 窗口中改变活动线程**

- 从线程列表中选择活动的线程来改变它。

有关窗口中指定元素的信息，见 Visual Studio 在线文档中的“Viewing the Call Stack”。

## 在 Running Documents（运行文档）窗口中查看信息

Running Documents 显示当前加载到正在运行的过程的文档列表。例如，如果添加一个 HTML 框架集（frameset）到项目中，Running Documents 窗口显示当前加载到浏览器中的是那一页。

**显示 Running Documents 窗口**

- 在 View 菜单上，单击 Debug Windows，然后选择 Running Documents。
  或者
  按下组合键 CTRL+ALT+R
  或者
  单击在 Debug 工具栏上的 Running Documents 按钮。

**从 Running Documents 窗口中打开文档**

- 双击文档在编辑器中打开它。

**在顶部文档中查看或隐藏文档**

- 在顶部文档中展开或折叠列表来查看或隐藏文档。

## 单步执行代码

用于此方案的应用程序示例接收从命令行传递的日期，并将其转换为 Julian 日历日期。在该方案中使用的应用程序在本书的第 1 章中的“创建控制台应用程序”一节中建立。如果想要创建该项目来完成下列练习，现在就开始做。

用户需要输入日期用于转换、停止在已经设置的断点上，并且使用调试窗口来检查程序中的一些变量。你可能知道问题的起因，但最好是单步执行代码中的语句，以便更好地理解实际发生的事情。下列过程显示了如何跟踪程序的执行。

### 在中断模式单步执行程序

1. 将光标放在所要开始单步执行代码的那行的页边上。

   **提示：** 选择在应用程序结束之前所要执行的行。

2. 如果使用下面列出的方法用来调试 Console 应用程序，将光标放在下面列出的代码旁边：

```
for (int nCount = 1 ; nCount < m_nMonth; nCount++ )
```

3. 在页边单击。

   一个红点出现在页边上，指出该行上设置了断点。

4. 按下 F5 键，调试器开始运行该应用程序并且中断在新设置的断点上。
   在页边上的黄色箭头指出将要处理的下一个指令。
5. 按 F11 键单步继续执行应用程序，注意代码中执行的行和打开的调试窗口中徐显示的信息。
   在打开的调试窗口中，只要成员变量的值改变，则以红色显示。如果正在接着调试用于 Console 应用程序的方案，则继续单步执行应用程序的源代码，直到该值以红色在打开的调试窗口显示，并且为 m_nDayOfYear 和 nCount 时为止。
   有关调试器对你的程序代码的分步调试的详细信息，请参见 Visual Studio 在线文档的“分步跟踪执行代码”。

**终止调试过程**

- 在 Debug 菜单上，单击 End（结束）。
  关于终止调试过程的更多方法，见在 Visual Studio 在线文档中的“Ending a Dubugging Session”。

当程序进入断点模式时，用户也许想要设置其他断点，或者检查一个和多个 Debug Windows 来查看程序的成员变量值。
有关使用集成调试器的其他信息，见本章前面的“基本调试过程”一节。

## 调试WFC应用程序

如果已经创建了一个Windows Foundation Classes for Java(WFC)应用程序，同时注意到了它们在运行时的特殊行为，下面的过程将帮助用户调试这样的应用程序。

> **注意：**用于该方案的应用程序示例在第 1 章中的“使用 WFC 创建 Windows 应用程序”一节中建立。如果想要创建该项目来完成下列执行操作，现在就开始做。

在该方案中的应用程序接收一个日历日期，并且将其转换为一个 Julian 日期。一旦日期已经输入，用户将设置断点并开始调试该应用程序。用户还使用调试器窗口检查一些成员变量的值。尽管可能涉及到在 Julian Data Conversion 应用程序中特殊的代码语句，用户还是能够从应用程序中替换代码，并且在这个有用的方案中始终找到该过程。

当输入了要转换为 Julian 日期的日期时，参照本章前面的“在启动调试器之前设置断点”一节来开始调试过程。

当应用程序的执行在一个断点上中止时，在 Auto，Locals，Watch 或 Immediate 调试窗口中查看用在应用程序中的成员变量的当前值，这便可以知道在程序进入断点模式之前时的瞬间状态。

现在，可以看看程序运行到断点模式时所处的状态了，用户可能想要设置其他的断点，并且继续单步执行代码。

这些就是调试 WFC 应用程序的方案。有关在使用集成调试器方面的其他信息，见在本章前面的“基本调试过程”一节。

## 调试控制台应用程序

如果已经创建了一个 Java 控制台应用程序，并且注意到了它们在运行时的特殊行为，下面的过程将帮助用户调试这样的应用程序。

> **注意：**用于该方案的应用程序示例在第一章中的“创建控制台应用程序”一节中建立。如果想要创建该项目来完成下列执行操作，现在就开始做。

在该方案中的应用程序接收从命令行传递的日历日期，并且将其转换为一个 Julian 日期。一旦日期已经输入，用户设置断点并开始调试该应用程序。用户还使用调试器窗口检查一些成员变量的值。尽管可能涉及到在 Julian Data Conversion 应用程序中特殊的代码语句，用户还是能够从应用程序中替换这些代码和成员变量，在这个方案中的过程仍是有用的。

运行该应用程序需要在该程序运行之前从命令行中输入一个日期。在过程中的这一步参见在本章前面的“输入命令行参数”一节，在该节中显示了如何从开发环境中的 Launch 选项卡中传递该信息到应用程序中。当为程序输入了命令行参数之后，参照本章前面的“在启动调试器之前设置断点”一节来开始调试过程。

当应用程序的执行中止在一个断点上时，在 Auto、Locals、Watch 或 Immediate 调试窗口中查看应用程序中成员变量的当前值，以便了解在程序进入断点模式之前的位置时的瞬间状态。

现在，可以看看程序运行到断点模式时所处的状态了，用户可能想要设

置其他的断点，并且继续单步执行代码。

这些就是调试一个使用 Julian Date Conversion 应用程序的应用程序的方案。有关在使用集成的调试器方面的其他信息，见在本章前面的“基本调试过程”一节。

## 调试多线程应用程序

当一个线程是在进程中连续执行的代码时，进程是虚拟内存空间、代码、数据和系统资源的集合。处理器执行线程，而不是进程，所以每个 32 位应用程序至少有一个进程和一个线程。在引入执行多线程之前，应用程序全部设计为执行单线程。

如果已经创建了一个多线程应用程序或组件，并且注意到了当其运行时的特殊行为，用户可能需要在指定的线程上调试代码的运行，从而找出错误的根源。该解决方案的过程设计为显示如何使用一起调试 Threads 窗口和设置断点，并且在其他的调试窗口中查看信息。

如果没有支持多线程的应用程序，并且还想要执行下面的进程，完成在下一节“多线程 Beverages 应用程序”中介绍的创建多线程控制台应用程序的操作。

## 多线程Beverages应用程序

下面列出的方法用来创建一个支持两个线程的控制台应用程序。该应用程序由3个类组成：Class1，Coffee和Tea。Class1包含应用程序的main()方法和代码，用来启动两个线程——运行Coffee类和用于Tea类的代码。

### 创建多线程控制台应用程序

1. 在File菜单上，单击New Project来显示New Project对话框。
2. 展开Visual J++文件夹，然后展开Application文件夹。
3. 单击Console Application图标。
4. 在Name框中，为项目输入唯一的名称。
5. 在Location（位置）文本框中，输入或浏览到想要保存项目的位置。
6. 单击Open，并且项目出现在Project Explorer中。
7. 在Project Explorer中，展开项目的图标，并且在Class1.java文件上双击。

现在所创建的.java文件已经加载到Text编辑器中，并且准备进行修改。

### 将Coffee和Tea类添加到Class1.java文件中

1. 在文件的结尾，Class1的结束大括号“{”后面，输入在下一部分“Coffee和Tea的源代码”中的代码。
2. 在Class1的main()方法的两个大括号之间，插入下列代码：

```
Coffee m_Coffee = new Coffee() ;          // creates Coffee object
```

```
m_Coffee.start();                    // creates thread for Coffee object
new Tea().start();                   // creates Tea object and its thread
```

## Coffee 和 Tea 的源代码

下面的代码段支持多线程 Beverages 应用程序的创建。下面列出的 Coffee 和 Tea 类展开允许它们在单独的线程上运行的 Thread 类。

```
Class coffee extends tread
{
    public void run()
    {
        for(int I = 0; I <10; I++)
        {
            System.out.println("I Like Coffee" + " " + I);
            yield();
        }
    }
}
```

```
class Tea extends Thread
{
    public void run()
    {
        for(int I = 0; I < 10; I++)
```

```
            {
                System.out.println("I Like Tea" + " + i);
                yield();
            }
        }
}
```

## 调试多进程应用程序

当一个线程是在进程中连续执行的代码时，进程是虚拟内存空间、代码、数据和系统资源的集合。处理器执行线程，而不是进程，所以每个 32 位应用程序至少有一个进程和一个线程。在引入执行多线程之前，应用程序全部设计为执行单线程。进程与其他进程通过消息进行通信，使用 RPC 在进程之间传递信息。对于调用程序来说，来自在远程计算机上的进程的调用与在同一台计算机上其他进程的调用没有什么区别。

当正在调试要求多个进程来运行的应用程序时，用户可能想监视所有进程的活动。下面列出的过程显示了在调试应用程序时，如何选择不同的进程。

### 从 Threads 调试窗口中选择进程

1.打开 Threads 调试窗口。

2.从 Threads 窗口顶部的下拉式列表中，选择所要观察的进程。

一旦已经选择了一个进程来调试，见本章前面的“基本调试进程”一节。一些特殊的进程请参见下面列出的几部分：

- 显示 Debug Toolbar
- 设置断点
- 使用 Debug 窗口检查信息
- 单步执行代码
- 调试多线程应用程序

## 调试COM对象

Component Object Model（COM）服务器是一个动态链接库（DLL）或是实现一个和多个类，而类又实现一个和多个接口的可执行文件。COM接口是一个相关函数的组，并且，通过该对象支持的各种接口来完全访问该对象。

调试 COM 服务器与调试可执行文件或小应用程序之间有着很大的不同，主要是因为 COM 服务器并不是用来作为单独的应用程序来运行。它的存在是用来为客户提供服务。这样的客户可以是可执行文件、Active Server Pages（ASP，活动服务器页面）或其他动态链接库。

所以，要调试 COM 服务器，用户应该将项目加载 Visual J++中，然后还需要将 Visual J++环境配置到启动调试客户。接下来，应该在对象代码中设置断点，这样，当 Visual J++环境启动调试客户时，对象代码的执

行将在指定的点上中止。当激活断点之后，用户可以单步调试服务器对象代码，就像正在调试一个可执行文件或小应用程序一样。
用来调试对象的方法主要取决于两个因素：将访问到对象服务的调试客户类型和对象被设计用来操作的环境。
这部分介绍三种调试 COM 服务器的方法。这些方法根据客户类型而不同，包括下述内容：

- 使用 Active Server Pages（ASP）调试客户。用于 Active Server Pages 环境中的特殊设计的对象用来作为服务器端的组件。这一节解释了如何调试一个服务器端使用 ASP 客户的组件。详细调试脚本命令参见在 Visual INterDev 在线文档中的“The Script Debugging Process”。
- 使用可执行文件调试客户。这一节解释了如何配置 Visual J++环境来启动将使用对象服务的可执行程序。
- 使用 Microsoft Transaction Server（MTS）。调试包括 MTS 自身和调试客户之间相互作用的 MTS 组件。这一节解释了如何调试启用 MTS 的组件。

## 使用 Active Server Pages（ASP）调试客户

在开始调试服务器端的组件之前，确定已经：

- 创建了一个 ASP 用来运行服务器端组件。

- 在计算机或 Web 服务器上已安装了 Microsoft Internet Information

Server（IIS），IIS 在计算机或 Web 服务器上配置用来调试 Java 服务器端组件。

- 将组件展开到 IIS。
- Internet Explorer（4.0 或更高版本）。

该方案使用在本书第 1 章中“创建 COM DLL”一节中创建的项目。但是，下面列出的过程提供了准备调试任何服务器端组件所需的信息。

## 准备用于调试 Java 服务器端组件的环境

**准备用于调试 Java 服务器端组件的环境**

1. 在 Tool 菜单上，单击 Options 并在 Options 对话框中的树形视图中突出显示 Debugger 节点。
   Debugger 选项的属性页出现。
2. 在该属性页的 Java 和 Script 部分中，选择 Attach to running program（附加到运行程序）选项。
3. 重新启动计算机来启用这些选项。
4. 启动 Internet Explorer，并且将浏览器指向在 http://服务器上项目已经展开 Java 服务器端组件的.asp 页的位置上。
   .asp 页的内容出现在 Web 浏览器上。

当已经配置了 IIS 和准备 Visual J++环境之后，现在已经做好了调试 Java 服务器端组件的准备。

# 启动 Java 服务器端组件调试进程

## 调试 Java 服务器端组件

1. 启动 Visual J++。
2. 在 File 菜单上，单击 Open Project（打开项目）。
3. 在 Open Project 对话框中的 Existing（已有项目）选项卡上，选择服务器端组件项目。
4. 在 Project Explorer 中，展开该项目的节点，并且双击组件的 .java 文件来将其加载到 Text 编辑器中。
5. 在 Debug 菜单上，单击 Processes（进程）来显示 Processes 对话框。
6. 在该对话框中的 Processes 部分，选择 Microsoft Active Server Pages。
7. 单击 Attach（附加）。
   此时使项目进入运行模式，并且启动调试器。
8. 从 Text 编辑器中，在组件类的公共方法中设置断点。
9. 从 Internet Explorer 中，在脚本调用在 .asp 页中的组件之后设置断点。
10. 从 Internet Explorer 中，单击 Refresh（刷新）。
    脚本代码调用该组件，并且在设置断点的行上中止。

现在，环境和项目已经做好了准备，可以使用调试器的窗口和设置断点的常规属性继续调试，单步执行代码，并且在错误后继续执行。

## 使用可执行文件调试客户

在服务器端组件项目内，可以指定和配置一个调试执行文件、在组件中触发断点的应用程序。

### 配置客户可执行文件

1. 在 Project 菜单上，单击 <*Project Name*>Properties（〈项目名〉属性），这里的项目名就是当前加载的项目名称。
2. 在 Properties 对话框的 Launch 选项卡中，单击 Custom（定制）。
3. 在 Program（程序）框中，输入调用到组件中的可执行文件的路径。
4. 在 Arguments（参数）框中，输入要传递到在 Program 框中指定应用程序中的参数。
5. 单击 OK。
6. 在组件的类代码中，将光标放到想要设置断点的位置上。
7. 在 Debug 菜单上，单击 Start 来启动作为调试客户指定的可执行文件。

当 Visual J++调试器在指定的断点中断了组件代码时，用户可以使用在调试器 Debug 菜单上的单步命令（Step Over、Step Into 和 Step Out）在组件的源代码移动。

## 使用 Microsoft Transaction Server（MTS）

如果设计的组件用来支持 MTS 事务，用户可以使用 MTS 和客户可执行文件来调试组件的代码。

**配置 MTS 调试**

1. 在 MTS 已经安装在其中的 Windows NT Server 上，关闭服务器进程。要做到这一点，在 Windows NT 桌面上右击 My Computer 来显示快捷菜单，然后单击其中的 Shutdown Server Process（关闭服务器程序）。
2. 在 Visual J++环境中，单击 Project 菜单，然后单击<*Project*>Properties（〈项目〉属性），这里的项目就是用户的组件项目的名称。
3. 单击 Launch 选项卡，然后单击 Custom。
4. 在 Program 框中，输入到 MTx.exe 的路径。默认情况下，可执行文件保存在 Windows NT System32 目录中。
5. 在 Arguments 框中，输入/p:“MTS 软件包名称”，这里的软件包名称是指包含用户组件的 MTS 软件包名称。还须确定在参数中没有空格。
6. 在组件代码中设置断点。
7. 在 Debug 菜单上，单击 Start。
8. 手工启动调用到组件中的可执行文件。当客户调用到已经设置断点的代码段时，执行中止，并且用户可以单步执行组件代码。

## 调试 Java 小应用程序

一旦在 JVIEW 或 Internet Explorer 中启动小应用程序，对于调试会话期间的特殊过程，见本章前面“基本调试过程”一节。

## 在 JVIEW 中调试小应用程序运行

环境和项目现在已经做好了准备，可以使用调试器窗口和常规特征来继续调试小应用程序，如设置断点、单步执行代码和在错误后继续执行代码。

## 在 Internet Explorer中调试小应用程序运行

环境和项目现在已经做好了准备，可以用调试器窗口和常规特征来继续调试小应用程序，如设置断点、单步执行代码和出错后继续执行。

# 第 7 章　打包和部署项目

当准备发布应用程序时，Visual J++开发环境包含所有需要的打包和展开项目的工具。
可以以下面列出的输出格式打包 Visual J++项目：

- COM DLL
- 自动扩展设置（.exe）
- Windows EXE
- Zip 归档文件（.zip）
- 小型归档文件（.cab）

当建立项目时，用户可以使用 Properties Project 对话框的 Output Format（输出格式）选项卡来指定所要创建的软件包的格式。
如果将项目作为小型归档文件或作为自动扩展安装文件打包，用户可以单击 Project Properties 对话框中的 Advanced（高级）按钮来签名该项目，以增加安全性（如果在这里不做改变，则.cab 文件将按照在 Options 对话框中的 Security（安全）部分中指定的默认身份鉴别设置进行签名）。
当准备发布项目时，Visual J++开发环境允许将项目部署到 Web 节点和 Web 服务器上，用来测试和生产。

**注意：**在使用 Application Wizard 创建项目时，用户可以设置有限的打包和部署选项。

有关打包和部署项目的详细信息，参见 Visual Studio 文档中的下列标题：

- Solution Building and Packaging Concepts
- Solution Building and Packaging User Interface Reference
- Solution Building and Packaging Common Tasks
- Solution Deployment Concepts
- Solution Deployment Common Tasks
- Solution Deployment User Interface Reference

# 第 8 章　使用源代码控制管理项目

Visual J++环境将源代码控制系统集成到 Project Explorer 及其他菜单和窗口中。用户可以使用源代码控制来控制源文件（项）的位置和版本，设置源代码控制项目层次来匹配开发目录树，并且通过延迟添加、删除和重命名源代码控制数据库来更有效地管理组项目。要学习有关使用完整的源代码控制系统来管理 Visual J++项目的知识，见 Visual Studio 在线文档中的下列主题：

- Source Code Control Concepts
- Source Code Control Common Tasks
- Source Code Control User Interface Reference

# 第 9 章　使用 JVC 编译应用程序

JVC.EXE 是用于 Java 的 Microsoft 编译器。默认情况下，编译器产生运行在任何用于 Java 的虚拟机上的（.class）文件。但是 JVC 还编译对于 Windows 平台来说最优化的应用程序。有关使用 Visual J++创建 Windows 应用程序的更多信息，见本书第 19 章。

用户可能在项目的 Properties 对话框中的 Compile（编译）选项卡上选择项目的编译选项；选项还可以输入到同一个选项卡的 Additional compiler options（附加编译器选项）文本框中。从 Project 菜单的 Properties 对话框中访问 Compile 选项卡。在“设置编译器选项”一节中说明了在 Compile 选项卡上的每个选项。

## JVC 语法说明

JVC.EXE（JVC）命令行使用下列语法：

JVC [*option*] <*filename*>

下面的表格说明了在 JVC 命令中的输入。

| 项目 | 含义 |
|---|---|
| option（选项） | 一个和多个 JVC 选项。有关更多的信息，见在本书第 1 |

| | |
|---|---|
| | 章中的“设置 JVC 选项”和本章后面的“JVC 命令行选项”部分<br>**注意**：所有选项将应用于所有指定的源文件。 |
| filename（文件名） | 一个和多个源文件的文件名。有关有效文件名的信息，见这一节后面的“文件名语法” |

用户可以指定任何数量的选项和文件名，但在命令行中字符的数量不能超过 1024 个字符或操作系统支持的最大长度。

> **注意**：不能保证 Microsoft Windows NT（版本 4.0 或更高）和 Microsoft Windows 95 将来发布的版本将都在命令行上有同样的 1024 个字符的输入限制。

所有想要传递到 JVC 的选项必须在.java 文件之前提供。

### 文件名语法

JVC 识别下面列出的文件名句法：

- JVC 接收符合 FAT 和 NTFS 命名约定的文件。
- 包含完全路径或部分路径中的文件名。
  完全路径包括驱动器名称和一个或多个目录名称。JVA 接收由反斜线符号（\）或正斜线符号（/）分开的正文件名。
  部分路径省略了驱动器名称，这样，JVC 认为文件在当前的驱动器中。如果没有指定路径，则 JVC 认为文件在当前的目录中。

**注意：**如果文件名没有带有扩展名，当 JVC 读该文件时，认为该文件的扩展名为 .java。

## 使用 JVC.EXE 编译

可以使用 JVC.EXE（JVC）来编译指定的 .java 文件到 .class 文件中，这个 .class 文件通过 Java 的虚拟机运行。如果只需要检查代码的语法错误而不产生 .class 文件，则使用 /nowrite 选项。

可以在命令行上或是在开发环境中指定 JVC 选项。如果正在从开发环境中编译 Java 项目，在 Compile 选项卡的 Additional compiler options 中指定想要使用的 JVC 选项。从 Project 菜单的 Settings（设置）对话框中访问 Compile 选项卡。

如果正在从命令行编译 Java 项目，将 JVC 选项放在命令行上的一个和多个 .java 文件名的前面。

### JVC 选项的顺序

命令行选项必须出现在 JVC.EXE 命令行上 .java 源文件名的前面。JVC 按照从左到右的顺序读取命令行，它也按照这个顺序来处理所遇到的选项。每个选项应用于在命令行上的所有文件中。如果 JVC 遇到了冲突选项，则使用最右边的选项。

## JVC命令文件

命令文件，也称之为响应文件，是一个包含想要另外在命令行上输入的文件名列表的文本文件。JVC 在命令行上接收命令文件作为参数。用命令文件可以编译多个源文件。
当使用命令文件时，应当符合下述条件：

- 命令文件包含一个文件名。
- 命令文件不能调用 JVC 命令。
- 命令文件不能包含 JVC 编译选项。

命令文件由文件名后面的“at”符号(@)来指定；文件名可以指定绝对或相对路径。

## CLASSPATH环境变量

CLASSPATH 环境变量指定系统默认的.class 文件的位置。当 Java 编译器（JVC）、应用程序查看器（JVIEW 和 WJVIEW）和虚拟机编译或运行 Java 程序时，使用存储在 CLASSPATH 环境变量中的信息。当从命令行编译时，使用/cp:p 和/cp:a 来添加信息到 CLASSPATH 环境变量中。

> **注意：**用于 JVC、JVIEW 和 WJVIEW 的选项只应用于当前编译，每次运行这些工具时都必须输入这些设置。

在命令行中设置 CLASSPATH 环境变量，则会将其改变，直到系统关闭

时为止。但是，在编译或查看 Java 程序时，JVC，JVIEW 和 WJVIEW 的/cp，/cp:p 和/cp:a 选项可以替换或编辑 CLASSPATH 环境变量。

**注意**：在 Microsoft Windows 系统上，CLASSPATH 环境变量中的目录由分号分开。

在命令行上，使用下列句法设置 CLASSPATH 环境变量：

SET CLASSPATH = <path>

## CLASSPATH 和 Java Package Manager（JPM，Java 打包管理器）

当安装 VM 时，它使用 Java Package Manager（JPM）注册有关所有 Java 软件包的信息，并将下列 CLASSPATH 值输入到 HKEY_LOCAL_MACHINE\Software\ Microsoft\Java VM\Classpath 注册表键中。

Windows 95　　C:\WIN95\java\classes\classes.zip;C:\WIN95\java\classes;.;
Windows NT
　　C:\WINNT\java\classes\classes.zip;C:\WINNT\java\classes;.;

当运行 Java 程序时，VM 在下列来源中按照列出的顺序收集信息，用以查找.java 文件：

1. 命令行提供的路径。

注意：当运行 JVC、JVIEW 和 WJVIEW 时，使用/cp，/cp:p 和/cp:p 选项来指定 CLASSPATH 信息。

2. DevClasspath 注册表键。

注意：利用 DevClasspath 注册表键，从事软件包的开发者可以取代可能在 Java Package Manager（JPM）的数据库中安装的软件包。使用这些键，开发者可以将它们的软件包编到指定的目录，并且 JPM 的安装软件包将不显示它。有关 Java Package Manager 的更多的信息，见 Java Package Manager 文档。

3. Java Package Manager（JPM）数据库。

注意：当程序从 Visual J++开发环境中运行时，VM 包括在查找.class 文件时输入到 Settings 对话框的 Java Packages 选项卡中的信息。从 Project 菜单的 Settings 对话框中访问 Java Packages 选项卡。有关 Java Package Manager 的更多信息，见 Java Package Manager 文档。

4. TrustedClasspath、TrustedLibsDirectory、Classpath 和 Libsdirectory 注册表键。
5. CLASSPATH 环境变量。

# JVC命令行选项

本节是所有JVC命令行选项的参考，按字母表顺序列出。这些选项在下面的表中给出。

如果命令行选项可以带一个和多个参数，则它的句法在语法标题下显示，在说明的前面。单击下表中的任何选项都可以查找该选项的信息。

**命令行中的JVC选项设置**

| | | | |
|---|---|---|---|
| /cp — 设置CLASSPATH选项 | /g—生成调试信息选项 | /nowrite—只编译选项 | /verbose—显示编译器信息选项 |
| /cp:o — 显示CLASSPATH选项 | /g:l—生成行编号信息选项 | /O—允许优化选项 | /w — 设置警告级别选项 |
| /cp: — 前加CLASSPATH选项 | /g:t—生成调试表选项 | /O;I—内嵌方法优化选项 | /x—禁止语言扩展选项 |
| /d—输出目录选项 | /nologo—禁止版权标志 | /O:J—优化字节码跳转选项 | /?—在线帮助选项 |
| /D — 定义条件编译符号 | /nowarn—禁止警告信息 | /ref—重新编译引用的类 | |

表中的所有选项都可以输入到Compile选项卡的Additional compiler

options 文本框中。从 Project 菜单的 Settings 对话框中访问 Compile 选项卡。这些选项还可以用来输入到命令行的源文件名之前。

## /cp——设置CLASSPATH选项（JVC）

### 句法

/cp *classpath*

使用/cp 选项来设置用于当前编译的 CLASSPATH 信息。使用该选项指定 JVC 能够找到系统和用户定义的类的路径。Java 的虚拟机使用 CLASSPATH 环境变量和 Java Package Manager 来查找系统类。有关使用 Java Package Manager 的更多信息，见在线文档 Microsoft SDK for Java 文档中的 Java Package Manager 文档。

> **注意：** 在 CLASSPATH 环境变量中的目录在 Microsoft Windows 系统上用分号分开。

### 例子

在 Windows NT（版本 4.0 或更高）中编译 myClass.java 时，类路径应该为：

```
JVC /cp x;.; x:\java\class myClass.java
```

在该例子中，JVC 在当前目录和 x:\java\class 目录下查找系统和用户定义的类。

## /cp:o——显示CLASSPATH选项（JVC）

该选项打印类路径到标准的输出设备。当从命令行进行编译，如果发生如“class not found（找不到类）”这样的错误时，该选项特别有用。

### 例子

下面的例子将类路径打印到屏幕：

```
JVC /cp:o
```

## /cp:p——前加CLASSPATH选项（JVC）

### 句法

```
/cp:p path
```

该选项前加在CLASSPATH环境变量中找到的信息，并且在目录路径中插入一个分号。当输入多个/cp:p选项时，路径连接。

> **注意：**CLASSPATH环境变量或/cp选项提供.class文件的位置。对于CLASSPATH环境变量的完全说明，见本章前面的“CLASSPATH环境变量”一节。

### 例子

下面的命令前加 myproj 的目录到已有的类路径中：

```
JVC /cp:p myproj
```

下列命令连接目录 myproj1 和 myproj2，并且前加结果路径到已有的 CLASSPATH 信息。

JVC /cp:p myproj1 /cp:p myproj2

## /d——输出目录选项（JVC）

### 语法

/d directory

当编译.java 文件时，使用/d 选项来为.class 文件指定一个在当前目录之外的输出目录。如果目录不存在，JVC 将创建它。如果该选项没有指定，JVC 将.class 文件写到包含相应的.java 文件的目录中。

### 例子

下面的命令编译 myClass.java 文件到 myClass.class 文件中，并且将该文件写到 classdir 目录中。

JVC /d c:\classdir myClass.java

> 警告：Java 的虚拟机根据包含.class 文件的 Java 软件包名称，依赖于.java 类的特定目录位置。使用该选项，如果 VM 不能找到它需要运行的 Java 程序的类时会导致执行失败。有关 VM 如何查找.class 文件的说明，见本章前面的“CLASSPATH 环境变量”一节。

## /D——定义条件编译符号

### 语法

/D *identifier*

该选项告诉编译器把标识符当作对所有编译文件上使用的条件编译的定义符号。该符号能够在.java 源文件中使用#if 伪指令来测试是否定义了一个符号。更多的信息见“条件编译”。

### 例子

下列命令使用 WINDOWS 和 DEBUG 预处理符号定义编译 myClass.java 文件：

```
JVC /D WINDOWS /D DEBUG myClass.java
```

## /g——生成调试信息

### 语法

/g [-]

/g 选项生成所有调试信息。默认不设置调试选项。使用/g 选项的效果同一起使用下列选项的效果一样。

| 选项 | 效果 |
| --- | --- |
| /g;l | 生成行编号信息 |

/g:t　　生成调试表

**注意：**在命令行上该选项后面使用破折号（—）来禁用该选项。

**例子**

下面的命令创建一个名为 myClass.class，并且包含调试信息的的.class 文件：

JVC /g myClass.java

## /g:l——生成行编号信息选项（JVC）

**语法**

/g:l[-]

该选项生成当调试程序时所使用的行编号。默认情况下不设置调试选项。

**注意：**在命令行上该选项后面使用破折号（-）来禁用该选项。

**例子**

下面的命令通知 JVC 为结果 myClass.class 文件生成行编号信息：

JVC /g:l myClass.java

## /g:t——生成调试表选项（JVC）

**语法**

/g:t[-]

这些选项生成调试程序时使用的调试表。默认时不设置调试选项。

**注意：**在命令行该选项后面使用破折号（-）来禁用该选项。

**例子**

下面的命令通知 JVC 来为结果 myClass.class 文件生成调试表信息。

JVC /g:t myClass.java

## /nologo——禁止版权标志

**语法**

/nologo

当从命令行编译文件时，该选项禁止显示编译器版本号和版权信息。

**例子**

下面的命令在不显示编译器的版权通知的情况下编译 myClass.java 文件。

JVC /nologo myClass.java

## 禁止警告消息

### 语法

/nowarn

该选项导致编译器不出现任何警告消息，而只出现错误消息。

### 例子

下面的命令在不显示任何警告消息的情况下编译 myClass.java 文件：

JVC /nowarn myClass.java

## /nowrite——只编译选项（JVC）

### 语法

/nowrite

/nowrite 选项通知 JVC 编译.java 文件并禁止写.class 文件。该选项在只检查源代码的语法错误，而不产生.class 文件时很有用。

### 例子

编译 myClass.java 文件，报告错误和警告，但不生成相应的 myClass.class

文件：

```
JVC /nowrite myClass.java
```

## /O——启用优化选项（JVC）

### 语法

/O

/O 选项结合优化选项来产生更有效率的程序。使用这些选项的效果与一起使用下面列出的选项效果相同：

| 选项 | 效果 |
|---|---|
| /O;I | 通过适当的内嵌方法优化 |
| /O:J | 优化字节码跳转。默认的最优化设置。 |

> 注意：在命令行该选项的后面使用破折号（-）来禁用所有的优化选项。

### 例子

下面的命令完全优化用于产生 myClass.class 文件的代码。

```
JVC /O myClass.java
```

## /O:I——内嵌方法优化选项（JVC）

### 语法

/O:I

/O:I 选项通知编译器它可能内嵌方法来产生更有效的代码。内嵌的代码没有与方法调用关联的开销。由于在 Java 语言中没有请求方法内嵌的机制，当想要编译器来内嵌代码时，使用该选项。

**注意：**在命令行该选项的后面使用破折号（-）来禁用该选项。

### 例子

下面的例子计算在 myClass.java 文件中的源代码，并且在可能的地方内嵌方法。结果 myClass.class 文件包含优化的代码。

JVC /O:i myclass.java

## /O:J——优化字节码跳转选项（JVC）

### 语法

/O:J

/O:J 选项使得 JVC 在编译.class 文件中优化字节码跳转。该选项通知编译器在代码中生成代码跳转到其他位置的代码。该选项是编译器的默认优化选项。

注意：在命令行该选项的后面使用破折号（-）来禁用该选项。

### 例子

下列例子计算在 myClass.java 文件中的源代码，并优化字节码跳转来产生更有效的代码。结果 myClass.class 文件包含优化的代码。

```
JVC /O:J myClass.java
```

## /ref　重新编译相关的类

### 语法

/ref [-]

该选项使得编译器在从类路径上的类文件中加载信息时，检查匹配的.java 源文件。如果设置该选项，无论何时编译器从一个.java 类文件中加载类信息，它都将检查在同一个目录中存在的与基本名相同的.java 源文件。如果找到这样的源文件，并且该源文件的修改日期比.java 类文件的要晚，则不读取.java 类文件，而是编译.java 源文件。

如果没有设置该选项，编译器将不管存在的源文件，而总是读取.java 类文件。

当从命令行开始编译时，该选项默认是打开的；使用/ref-来关闭该选项。

当从集成的开发环境中编译时，该选项默认关闭。

警告：在集成的开发环境中建议不要打开该选项，因为这样做可能会导致编译操作在设计时失败。

### 例子

下面的命令编译 myClass.java 文件，并且通知编译器不要重新编译任何相关的类。

```
JVC /ref- myClass.java
```

## /verbose——显示编译器消息选项（JVC）

### 语法

/verbose

该选项通知 JVC 在编译文件或项目时显示所有的消息。
这些消息提供了有关编译进展的有用信息。

## /w——设置警告级别选项

### 语法

/w0
/w1
/w2

/w 3

/w 4

/w x

这些选项控制由编译器产生的警告消息的输出。编译器根据警告的严重性来决定是否显示它们。这只影响在命令行上的文件名。

编译器警告消息的编号由 J5 开始。文件描述了这些警告，指出每个警告的级别，并指出可能的问题所在（比实际的代码错误要强一些）以及不是预期编译的语句。

这些选项的意义如下表所示：

| 选项 | 说明 |
|---|---|
| /w 0 | 关闭所有的警告消息 |
| /w 1 | 显示服务器警告信息 |
| /w 2 | 显示比/w 1 级别低一些的服务器警告信息。这是默认的警告级别设置 |
| /w 3 | 显示比/w 2 级别低一些的服务器警告信息，包括：使用带有未声明的返回类型的方法，将返回语句放入有返回值类型的方法中失败，导致数据或精度丢失的数据转换等 |
| /w 4 | 显示最低服务器级别的警告消息 |
| /w x | 将警告强制为错误。所有警告将导致编译器出现错误，并造成编译失败。该选项可以组合在上面的任何一种警告级别选项使用 |

## /x——禁止语言扩展选项（JVC）

### 语法

/x [-]

Visual J++编译器支持 Java 语言规范。另外，它可能提供大量的这些规范所需要的特性。这些特性默认是有效的，但是指定了/x 选项之后，这些特性不再有效。

如果用户计划将程序转换到其他环境中时，使用/x 选项。/x 选项通知编译器将扩展的关键字当做简单的标识符，并且禁用其他 Microsoft 扩展。

**注意：**在命令行上该选项后面使用破折号（-）来禁用该选项。

### 例子

下面的命令忽略任何用在 myClass.class 文件中的 Visual J++扩展。

JVC /x myClass.java

## /?——在线帮助选项（JVC）

### 语法

/?

该选项将编译器选项的列表显示到标准的输出设备上。

# 第 10 章　使用 JVIEW 和 WJVIEW 查看应用程序

## 使用JVIEW查看应用程序

JVIEW.EXE（JVIEW）是用来查看 Java 应用程序和小应用程序的工具。JVIEW 提供了 Java 程序可以运行的控制台窗口。JVIEW 支持应用程序和小应用程序的调试和销售版本。

如果用户的 Java 项目是一个应用程序，也就是说，具有一个 public static void main（String args[]）方法，JVIEW 将不需要附加的命令行选项，而在 JVIEW 控制台窗口中运行该应用程序。如果 Java 项目是一个小应用程序，也就是说，有一个 public void init()方法，则使用 JVIEW 的/a 选项。在本章后面的“JVIEW 命令行选项”一节中，以字母顺序列出 JVIEW 选项的参考。

## JVIEW 语法说明

JVIEW.EXE（JVIEW）命令行使用下面的语法：

JVIEW [options] <classname> [arguments]

下表描述了 JVIEW 命令行中的输入。

| 输入 | 含义 |
| --- | --- |
| Options（选项） | 一个和多个 JVIEW 选项，更多的信息见本章后面的“JVIEW 命令行选项”一节 |
| Classname（类名称） | 要执行的类名称。文件名中不包含.class 扩展名。例如，使用 HelloWorldApp，而不是 HelloWorldApp.class<br>**注意**：用户还可以指定一个全限定的、没有.class 扩展名的类名称，如 C:\My Documents\visual Studio Projects\ HelloWorldApp\HelloWorldApp |
| Arguments（参数） | 用来传递到由类名称标识的类中的命令行参数 |

**注意**：想要提供给 JVIEW 的任何选项必须在.class 文件名之前提供，或者它们将中断.class 文件的命令行参数。

## 运行 JVIEW.EXE

用户可以在命令行或在开发环境中指定 JVIEW.EXE（JVIEW）选项。默认情况下，Visual J++选择 WJVIEW 作为它的应用程序或小应用程序的查看器。在 Visual J++中，不指定附加的命令行参数而要运行 JVIEW，在 Launch 选项卡的 Default（默认）组中选择 Launch as a console application(作为控制台应用程序启动)复选框。从 Project 菜单的 Settings 对话框中访问 Launch 选项卡。

如果正在从命令行或从开发环境中运行程序，用户还可以将多于默认的选项传递到 JVIEW。

**从 Visual J++中传递多于默认的选项到 JVIEW**

1. 从 Project 菜单中，选择 Settings。
2. 在 Settings 对话框中的 Launch 选项卡中，选择 Other executables（另外执行）组框。
3. 输入在 other executables 组框中的 Program（程序）文本框中输入 JVIEW.EXE。
4. 将用于程序的 JVIEW 的选项输入到 Other executables 组的 Arguments（参数）文本框中。

当从命令行中运行 JVIEW 时，在命令行上类文件名称的前面输入想要传递到 JVIEW 中的选项。

## JVIEW命令行选项

本节是所有 JVIEW 命令行选项的参考，按字母顺序类出。这些选项在下面的表中给出。
如果命令行选项可以带一个和多个参数，则它的句法在语法标题下给出，在它的说明前面。

JVIEW 命令行选项

/a—小应用程序查看器选项　　/d:—系统属性选项

/cp—设置 CLASSPATH 选项
/cp:a—追加 CLASSPATH 选项
/cp:p—前加 CLASSPATH 选项
/p—暂停查看器选项
/v—类验证选项
/?—在线帮助选项

有关从开发环境和命令行中怎样指定这些选项的信息，参见本章前面的“运行 JVIEW.EXE”部分。

## /a——小应用程序查看器选项（JVIEW/WJVIEW）

使用/a 选项来调用一个小应用程序查看器——查看小应用程序的无浏览器的环境。运行的小应用程序查看器中的小应用程序与它们在查看器中的行为表现一样，包括加载、声音和安全性等。小应用程序查看器接受小应用程序参数和类名称，另外还有 URL 和 HTML 文件名。
/a 选项指示 JVIEW 或 WJVIEW 来运行一个小应用程序，并且需要 Java 项目中有一个 public void init()方法。如果用户的 Java 项目中有一个 public static void main(String args[])方法，使用不带有/a 选项的 JVIEW 或 WJVIEW 将其作为应用程序来运行。

> **注意：**当使用/a 选项时，在命令行上保留的任何符号都作为小应用程序查看器的参数。

### 例子

下面的命令行语句分别在 JVIEW 或 WJVIEW 的小应用程序查看器中运行 helloWorld 小应用程序：

JVIEW /a HelloWorld

或

WJVIEW /a HelloWorld

用户还可以通过使用HTML文件来指定参数或通过直接从命令行加载参数的方法加载一个小应用程序。下面的例子显示了如何从带有HelloWorld小应用程序选项卡的HTML源文件中加载这个小应用程序：

```
< applet code = MyApplet.class width = 100 height =200>
                <param name = SomeName value = SomeValue > < /applet >
```

同样，HelloWorld小应用程序可以在带有下面列出信息的JVIEW或WJVIEW中运行：

JVIEW /a width = 100 height = 200 SomeName = SomeValue HelloWorld

或

WJVIEW /a width = 100 height = 200 SomeName = SomeValue HelloWorld

下面的例子显示了如何从命令行同时运行多个小应用程序：

JVIEW /a width = 200 height = 400 HelloWord Height = 300 SpinningWorld

或

WJVIEW /a width = 200 height = 400 HelloWord Height = 300 SpinningWorld

前面命令的结果在一个 200×400 画面中显示 HelloWorld 应用程序，并且在一个 200×300 的画面中显示 SpinningWorld 小应用程序。

> **注意：**如果高度和宽度没有指定，则小应用程序默认的大小为屏幕尺寸的 3 分之 1。

包括 HTML 文件在内的所有参数将从 CLASSPATH 信息和 Java Package Manager（JPM）数据库所指定的目录和.zip 文件中加载，例如：

```
JVIEW /a /cp \src\bvt\HelloWorld.html
```

或

```
WJVIEW /a /cp \src\bvt\HelloWorld.html
```

## /cp——设置CLASSPATH选项（JVIEW/WJVIEW）

### 语法

```
/cp classpath
```

使用/cp 选项来设置用于当前编译的 CLASSPATH 环境变量。使用这些选项指定 JVIEW 和 WJVIEW 能够找到系统和用户定义的.class 文件的路径。Java 的虚拟机使用依赖于默认位置的平台，CLASSPATH Environment Variable（CLASSPATH 环境变量）和 Java Package Manager（JPM）数据库来查找系统类。有关 Java Package Manager 的更多信息，见在 Microsoft SDK for Java 文件中的 Java Package Manager 文件。

注意：在 Microsoft Windows 系统上，在 CLASSPATH 环境变量中的目录是由分号分开的。

例子

在 Windows NT 上，下面的命令提供用于 JVIEW 和 WJVIEW 的 CLASSPATH 信息：

JVIEW /cp X:.; X:\WINNT\java\classes\classes.zip;X:\WINNT\java\classes\

或

WJVIEW /cp X:.; X:\WINNT\java\classes\classes.zip;X:\WINNT\java\classes\

在该例子中，JVIEW 和 WJVIEW 在当前的目录下和 x:\java\classes 目录下查找系统和用户定义的类。

注意：在前面的例子中，X:指出了类或.zip 文件存在的驱动器符。

## /cp:a——追加CLASSPATH选项（JVIEW/WJVIEW）

语法

/cp:a path

/cp:a 选项将所输入的路径追加到 CLASSPATH 信息的末尾，并且在两个目录中间插入一个分号。当输入多个/cp:a 选项时，路径被连接。

**注意：** CLASSPATH 环境变量或/cp 选项提供.class 文件或.zip 文件目录的位置。Java 的虚拟机使用依赖于默认位置的平台，CLASSPATH Environment Variable（CLASSPATH 环境变量）和 Java Package Manager（JPM）数据库来查找系统类。有关 Java Package Manager 的更多信息，参见在 Microsoft SDK for Java 文件中的 Java Package Manager 文件。

**例子**

下面的命令追加 myproj 目录到已有的 CLASSPATH 信息：

```
JVIEW /cp:a myproj
```

或

```
WJVIEW /cp:a myproj
```

下面的命令连接目录 myproj1 和 myproj2，并且将结果路径追加到已有的 CLASSPATH 信息：

```
JVIEW /cp:a myproj1 /cp:a myproj2
```

或

```
WJVIEW /cp:a myproj1 /cp:a myproj2
```

## /cp:p——前加CLASSPATH选项（JVIEW/WJVIEW）

### 语法

/cp:p path

/cp:p选项前加所输入的路径到CLASSPATH信息的前面，并且在两个目录中间插入一个分号。当输入多个/cp:p选项时，路径连接起来。

> **注意：** CLASSPATH环境变量或/cp选项提供.class文件或.zip文件目录的位置。Java的虚拟机使用依赖于默认位置的平台，CLASSPATH Environment Variable（CLASSPATH环境变量）和Java Package Manager（JPM）数据库来查找系统类。有关Java Package Manager的更多信息，见在Microsoft SDK for Java文件中的Java Package Manager文件。

### 例子

下面的命令前加myproj目录到已有的CLASSPATH信息：

JVIEW /cp:p myproj

或

WJVIEW /cp:p myproj

下面的命令连接目录myproj1和myproj2，并且将结果路径追加到已有的CLASSPATH信息：

JVIEW /cp:p myproj1 /cp:p myproj2

或

WJVIEW /cp:p myproj1 /cp:p myproj2

## /d:——系统属性选项（JVIEW/WIVIEW）

**语法**

/d:property = string value

使用/d:选项来设置用于 Java 程序的系统属性。Java 程序通过使用 Java.lang.system. getProperties 方法可以读取系统属性。见这些方法中对于系统属性的说明。

**例子**

下面的命令将 user.dir 属性设置为任意值：

JVIEW /d:user.dir = c:\java\test HelloWord

或

WJVIEW /d:user.dir = c:\java\test HelloWord

用户还可以设置用户定义的属性。下面的命令设置名为“myprop”的属性：

JVIEW /d:myprop = 12 HelloWorld

或

WJVIEW /d:myprop = 12 HelloWorld

## /p——暂停查看器选项（JVIEW/WJVIEW）

/p 选项在 JVIEW 和 WJVIEW 出现错误时，在退出之前强制其暂停。在调试应用程序时，用户可以使用该选项来确定在出现错误之前的用户界面状态。

### 例子

使用下面的命令可以强制在 HelloWorld 应用程序发生错误并退出 JVIEW 或 WJVIEW 之前暂停。

JVIEW /p HelloWorld

或

WJVIEW /p HelloWorld

## /v——类验证选项（JVIEW/WJVIEW）

/v 选项使得 JVIEW 和 WJVIEW 验证所有调用的方法。如果不使用该选项，则只验证来自不信任的加载程序的方法。验证是应用到从.class 文件

中加载的字节码的进程，用于确保不会造成安全性的威胁。如果用户的某个类试图进行一些违反安全模式的操作时，验证可强制拒绝该文件。

> **注意：**在 Microsoft Java 的虚拟机上，只有该选项的全部功能的子集是可能带有用于远程加载的.class 文件的字节码语言。

因为验证要影响性能，所以本地系统类通常不验证。/v 选项强制验证程序来处理这些文件。用户可以使用这些选项来确保.class 文件集将通过验证程序。

**例子**

使用下面的 JVIEW 或 WJVIEW 命令来确保 SpinningWorld 应用程序的.class 文件不会造成安全性的问题：

```
JVIEW /v SpinningWorld
```

或

```
WJVIEW /v SpinningWorld
```

## /?——在线帮助选项（JVIEW/WJVIEW）

**语法**

/?

该选项显示一个 JVIEW 和 WJVIEW 的命令行选项的列表。

**例子**

使用下面的命令来分别显示 JVIEW 和 WJVIEW 的在线帮助：

```
JVIEW /?
```

或

```
WJVIEW /?
```

## 使用 WJVIEW 查看应用程序

WJVIEW.EXE（WJVIEW）是在命令行中用来查看基于 Windows 的应用程序的工具。WJVIEW 与 JVIEW 不同，它产生一个单独的窗口进程。WJVIEW 为用户基于 Windows 的程序提供一个可运行的环境，并且支持应用程序的调试和销售版本。

> **注意：** 使用 WJVIEW 来运行基于 Windows 的 Java 应用程序。WJVIEW 具有与 JVIEW 同样的功能和命令行选项。WJVIEW 在单独的图形用户界面（GUI）中运行基于 Windows 的 Java 应用程序，并且不使用或请求控制台窗口。因此，不要使用 WJVIEW 查看需要从控制台窗口输出或输入的 Java 应用程序。

WJVIEW 选项按字母顺序的参考，见本章后面的“WJVIEW 命令行选项”一节。

## WJVIEW 语法说明

WJVIEW.EXE（WJVIEW）命令行使用下面的语法：

WJVIEW [options] <classname> [arguments]

下表描述了 WJVIEW 命令行中的输入。

| 输入 | 含义 |
| --- | --- |
| Options（选项） | 一个和多个 WJVIEW 选项，更多的信息参见本章后面的“JVIEW 命令行选项”一节 |
| Classname（类名称） | 要执行的类名称。文件名中不包含.class 扩展名。例如，使用 HelloWorldApp，而不是 HelloWorldApp.class<br>**注意**：用户还可以指定一个全限定的、没有.class 扩展名的类名称，如 C:\My Documents\Visual Studio Projects\HelloWorldApp\HelloWorldApp。 |
| Arguments（参数） | 用来传递到有类名称标识的类中的命令行参数 |

## 运行 WJVIEW.EXE

用户可以在命令行或在开发环境中指定 WJVIEW.EXE（JVIEW）选项。

默认情况下，Visual J++选择 WJVIEW 作为它的应用程序或小应用程序的查看器。
如果正在从命令行或从开发环境中运行程序，用户还可以将多于默认的选项传递到 WJVIEW。

**从 Visual J++中传递多于默认的选项到 WJVIEW：**

1. 从 Project 菜单中，选择 Settings。
2. 在 Settings 对话框中的 Launch 选项卡中，选择 Other executables（另外执行）组框。
3. 在 Other executables 组框中的 Program（程序）文本框中输入 WJVIEW.EXE。
4. 将用于程序的 WJVIEW 的选项输入到 Other executables 组的 Arguments（参数）文本框中。

当从命令行中运行 WJVIEW 时，在命令行上类文件名称的前面输入想要传递到 WJVIEW 中的选项。有关指定选项的细节，见本章前面的“WJVIEW 命令行选项”一节。

## WJVIEW命令行选项

本节按字母顺序列出所有 WJVIEW 命令行选项。这些选项在下表中给出。
如果命令行选项可以带一个和多个参数，则它的句法在语法标题下给

出，在它的说明前面。

### WJVIEW 命令行选项

/a—小应用程序查看器选项
/cp—设置 CLASSPATH 选项
/cp:a—追加 CLASSPATH 选项
/cp:p—前加 CLASSPATH 选项
/d:—系统属性选项
/p—暂停查看器选项
/v—类验证选项
/?—在线帮助选项

有关在开发环境或在命令行上如何指定这些选项的信息，见在本章前面的“运行 WJVIEW.EXE”一节。