第一部分　基　　础

本部分包含如下各章：

- 第1章： Borland Developers Studio 3.0 简介
- 第2章： Win32 和 Microsoft .NET 平台
- 第3章： Delphi 编程语言
- 第4章： .NET 上的 Delphi 语言
- 第5章： Delphi Win32 运行时库
- 第6章：可视化构件库的体系结构
- 第7章：窗体处理
- 第8章： 用 VCL 创建用户界面
- 第9章： Delphi .NET 运行时库与框架类库

第 1 章　Borland Developers Studio 3.0 简介

在像 Delphi 这样的可视化编程工具中，集成开发环境所扮演的角色有些时候甚至比编程语言更为重要。如果读者熟悉 Delphi 7 或者更早期的版本，就会发现 Delphi 2005 基于一个更新、更开放的体系结构提供了一个完全重写过的 IDE。对于过去习惯于使用 Delphi IDE 的程序员来说，熟悉这个新的 IDE 将需要花费一些时间，因为 Delphi IDE 从 Delphi 1 到 Delphi 7 从未发生过根本的变化。这也是本章将给已经使用过 Delphi 早期版本的程序员提供一些技巧和建议的原因。此外，本章还将介绍一些新增的特性，以及对新入门的读者来说不太了解或者还不太熟悉的一些传统特性。但是，由于篇幅有限，本章将只提供一些介绍性的材料，而无法成为一本关于怎样使用 Delphi IDE 每个特性的综合性指南。

如果读者是初学 Delphi 编程的程序员，请不要担心。Delphi 的集成开发环境（IDE）使用起来是相当直观的。Delphi 本身提供了一本参考手册和一本介绍 Delphi 应用程序开发的辅导材料。Delphi 2005 IDE 已被引进到 Borland 的 C# Builder 和用于 Microsoft .NET Framework 的 Delphi 8 中。这也是 Delphi 2005 中的 Borland Developers Studio（BDS）被标以版本 3.0 的原因。

Delphi 的版本

在开始深入研究 Delphi 编程环境的具体细节之前，我们先来重点了解两个关键的概念。首先，Delphi 并不只有一种版本，而是有许多种版本。其次，任何 Delphi 环境都是可定制的。由于这些缘故，本章示例中所使用的 Delphi 屏幕与读者在自己的计算机上所看到的 Delphi 屏幕可能不完全相同。下面是 Delphi 的各个当前版本：

- 专业版（Professional）是真正的入门级版本，主要针对需求有限的专业开发人员，特别是在数据库连接和 Web 开发方面需求有限的开发人员。该版本包括了 Delphi 的所有基本特性、数据库编程支持（不完全支持客户/服务器开发）、基本 Web 服务器支持（WebBroker）以及一些外挂工具。
- 企业版（Enterprise）主要针对的是企业应用程序的开发人员。该版本包括完整的数据库支持、所有 XML 和高级 Web 服务技术、国际化支持、3 层体系结构以及许多附件工具。本书中的部分章节将会介绍企业版中所独有的特性，并且笔者将设法指出这些部分，但相关特性的详细情况请参考 Borland 网站（www.borland.com/delphi）上的 Delphi Feature Matrix 文档。请注意，过去曾经出现过这样的情况：虽然提供更新程序，但 Borland 给较早期的版本已经添加了特性，因而使得笔者在这方面力求准确的目标变得非常难以实现。
- 设计师版（Architect）在企业版的基础上添加了对许多 Borland Application Lifetime Management（应用程序生存期管理，简称 ALM）工具（比如 StarTeam 和 Caliber）的附加支持，以及对 Enterprise Core Object（企业核心对象，简称 ECO）的附加支持。ECO 框架是一种开发环境，用于开发在运行时由 UML 模型驱动的应用程序，

以及由于其中包含了丰富的高级构件而能将其对象映射到数据库和用户界面的应用程序。关于 ECO的详细描述，请参见本书的第 18 章。

注意：在 Delphi 的早期版本中，还有一个针对 Delphi 初学者的个人（Personal）版本，该版本在特性集方面是有限的（比如缺少数据库编程支持），并且在分发用它所编写的应用程序的权力方面也有限制。过去，这要么是一个免费（或者非常便宜）的下载版本，要么与杂志捆绑在一起。在编写本书的时候， Delphi 2005 的 Personal 版本还没有发布，而且也不清楚 Borland是否会开发 Delphi 的个人版本。

除了这些现有的不同版本之外，还有一些定制 Delphi 环境的技巧。在全书的屏幕图中，笔者尽量使用一个标准的用户界面（按照它们原来的样子）。可是，笔者当然也有自己的一些偏好，通常会安装许多附加工具，这有可能会反映到书中的屏幕图上。

IDE 概述

在简单了解了 Delphi 的各种版本之后，现在该是开始关注其集成环境（IDE）的时候了。即使对经验丰富的 Delphi 程序员来说，仍有许多需要介绍的东西，因为 Delphi 2005 IDE 有大量的新增特性。

IDE 的多重个性

除了有多种版本之外，Borland Developers Studio IDE 还有多重个性。Borland 使用术语“个性”来说明单个 IDE 既能用于处理不同的编程语言，又能以不同的平台为工作目标。换句话说，程序员不必运行一个用于 Win32 开发的 IDE，又运行另一个用于 .NET 开发的 IDE，而是可以运行单个 IDE并打开不同类型的项目，也许还能够创建由不同类型的项目所组成的项目组并利用单条命令整个地编译它们。

此外，Delphi 2005 有 3重个性：

- Delphi for Win32 个性：允许程序员持续开发用于标准 Win32 平台的程序。
- Delphi for .NET 个性：允许程序员将已有的 Delphi VCL 应用程序转移到 .NET 体系结构，或者编写特定的 WinForm 和 ASP.NET应用程序。
- C#个性：仅适用于 .NET，并基于 Microsoft 的编译器，但本书不讨论这一个性。

为了指出程序员正在使用的当前个性， IDE 在一个特定的工具栏中显示一个小图标（单击它将打开 About 框）。

IDE 是十分灵活的，不仅 Borland 可能会添加更多的个性，第三方开发者可能也会添加另外的个性（如果 Borland 愿意更多地透露一些关于这个过程的内部信息）。在发布 Delphi 2005 之后不久，Borland 宣称他们有意图让基于 VCL 的老式 C++ Builder 作为一个附加的个性，尽管在编写本书的时候他们还没有制定任何时间表和任何特性集。

注意：另一个可预见的选择是集成一个 Delphi for Linux 个性，这一个性也是读者几乎可以使用 Simon Kissel的 CrossKylix 工具来实现的东西。

一个用于 .NET和 ALM 的全新 IDE

读者可能很奇怪，Borland为什么选择开发一个全新的 IDE，而不是选择简单地升级

已有的 IDE。原因可能有许多，其中包括这样一个事实： Delphi 7 IDE 项目源于早期的 Delphi 而现在该是对这个用户界面进行彻底翻新的时候了，也许是因为有了一个更好的内部体系结构。

当然，设计一个新 IDE 的核心原因之一是需要在保持 Win32 特性的同时又完全支持 .NET 尤其是，这个 IDE 因集成 .NET SDK 设计器（比如 WinForm 设计器和 ASP.NET 设计器）而宿主了 .NET 运行时引擎。然而，其他 IDE 特性也依赖于 .NET。例如，所有的再加工支持（在第 11 章中加以讨论）都基于 CodeDOM 体系结构，而这个结构是 .NET Framework 的一个组成部分。

设计一个新 IDE 的另一个驱动原因是 Borland 作为一家公司有了发展。现在，Borland 不是将重点放在用来编写应用程序的工具上面，而是关注包括整个软件开发过程的工具：从需求分析到 UML 设计，从开发到优化、部署和测试。这一技巧早期称为 Application Lifetime Management（ALM），现在已演化并发展成一个策略，Borland 将其称为 Software Delivery Optimization（软件交付优化，简称 SDO）。

不考虑这个全局名称，对 Delphi 开发人员有重大意义的是，这个全新的 IDE 被设计成能够宿主由 Borland 内的不同部门所开发的许多其他工具。事实上， Delphi 2005 的 Architect 版本包括如下工具：

- StarTeam 客户：这个工具利用便于开发人员协作和提供与 ALM 套件的所有其他工具相集成的强大工具，不但支持版本管理，而且支持部署过程和产品的全面管理。
- Caliber 客户：这个工具用来管理需求。
- Together UML 建模工具：这个工具支持 ECO 框架，并且可以部分地用来开发和维护自定义 Delphi 代码的 UML 图。
- OptimizeIt：这个工具用来检查 .NET 应用程序中的瓶颈。
- 一个用于 ASP.NET 和 IntraWeb 应用程序的 Web Deployment Wizard（Web 开发向导）。

本书的重点是 Delphi，因此笔者将略微谈及这些 ALM 题目，因为其中的部分题目要求详细描述才能理解（仅仅几页的篇幅是没有用的），而且在本书将要重点讨论的专业版（Professional）和企业版（Enterprise）中没有包含这些题目。

不管怎样，都需要注意 IDE 主要是用 Delphi 本身编写的，只有某些部分是用其他语言编写的。另外需要特别注意的是，这些新 VCL 构件中的大多数已被添加进来，因为 Borland 在这个 IDE 自身内也需要它们（原因将在第 6 章中加以讨论）。

部分安装

Delphi 带有 3 重个性。但是，如果不需要全部使用它们（例如，不关心 C# 或者不再需要 Win32 支持），那么可以只安装这 3 个个性中的部分个性，也可以在事后删除它们。在编写本书的时候，据报道，部分安装可能会增加 IDE 的不稳定性，因此读者可能仍选择安装全部，即使不需要使用全部特性。

需要注意的是，BSD IDE 使用按需加载，这就意味着它在启动时仅加载每个个性的核心，因而将其他所需汇编的加载推迟到程序员实际开始使用某个指定特性的时候（比如程序员第一次处理一个 VCL 应用程序的时候）。这意味着程序员安装但未使用的个性不会给 IDE 运行时引擎增加大量负担。

即使程序员仅安装了 Delphi Win32个性，IDE 的某些特性仍需要依赖 .NET（比如前面曾经提过的再加工）。从技术上说，在一台没有安装 IDE运行时引擎的计算机上，将 Delphi 2005 用于 Win32 开发仍是可行的，但仅在程序员选择禁用那些特性的时候。

最后需要注意的是，程序员也可以完全禁用 .NET支持，也就是说，在计算机上不安装.NET 就使用 Delphi 2005。这并没有得到 Borland 的官方支持，但看起来像是行得通的，尽管笔者并不赞成读者这么做，因为再加工是笔者最喜爱的 Delphi 特性之一。另外，这个操作似乎使帮助系统完全不可用。然而，如果读者有兴趣，可以转到 http://delphi2005.cjb.net 网站上看一看这方面的详细说明。

能够删除由 IDE使用的某些程序包和程序组件在任何情况下都十分重要，假如读者知道自己将不再使用那些特性。例如，可以删除 Together或 StarTeam 支持，以便节省一些程序装载时间。下面是需要检查的 Windows注册表项目（责任自负，因为编辑注册表可能会在系统上产生麻烦）：

```
HKEY_CURRENT_USER\Software\Borland\BDS\3.0\Known IDE Assemblies
HKEY_CURRENT_USER\Software\Borland\BDS\3.0\Known IDE Packages
```

用多种配置启动 IDE

应当说明的一点是，给 Delphi 2005 设置不同的注册表项目集也是可能的，以便在每次执行时借助于一个命令行参数来选择要使用哪个注册表项目集。这意味着程序员可以创建 Delphi 的一个剥离版本，但仍让完全版本保持待命状态。

Delphi 有一个未正式公布的命令行参数：-r，从技术上说，程序员可以用这个参数指定要使用的基本注册表项目。例如，创建一个如下所示的快捷键(假设程序员有一个默认安装路径)：

```
"C:\Program Files\Borland\BDS\3.0\Bin\bds.exe" -rSmall
```

当程序员第一次运行它时， Delphi 利用来自下面这行代码的默认设置生成一个全新的注册表键集：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Borland\BDS\3.0
```

并将它们复制到：

```
HKEY_CURRENT_USER\Software\Borland\Small\3.0
```

换句话说，程序员所提供的名称将取代注册表树中的 BDS 名称。如果程序员不想用全新的设置进行启动，但希望编辑当前的配置，那么需要导出标准的 Borland \ BDS \ 3.0 注册表键值，并通过编辑它来修改键值名称，然后将它重新导入到注册表中。

除了这个 r参数之外（它在 Delphi 的早期版本中一直行得通，但未得到正式公布，而且现在仍未得到正式提及），还有程序员可以在启动 Delphi 2005 时使用的其他一些命令行参数。例如-hm 参数（Heap Monitor）在标题栏中显示由 IDE 自身所安装的内存数量，并在空闲时更新这个信息。另一个有趣的参数是-ns（No Splash）:它停止 Splash屏幕的显示。

欢迎使用 Delphi 2005

在运行 IDE 的时候，首先看到的将是 Welcome 页面，如图 1.1所示。这个 Welcome页面是一个包含 Internet Explorer 浏览器的窗格，并允许查看一些重要信息，也允许浏览

Web。它有 4 个区域：工具栏的下方有一些代表常见任务（比如打开项目和文件）的按钮，左边有一个含有本地和在线链接的菜单，右边是一个含有最近项目的列表，再下面是 Headlines 区域。

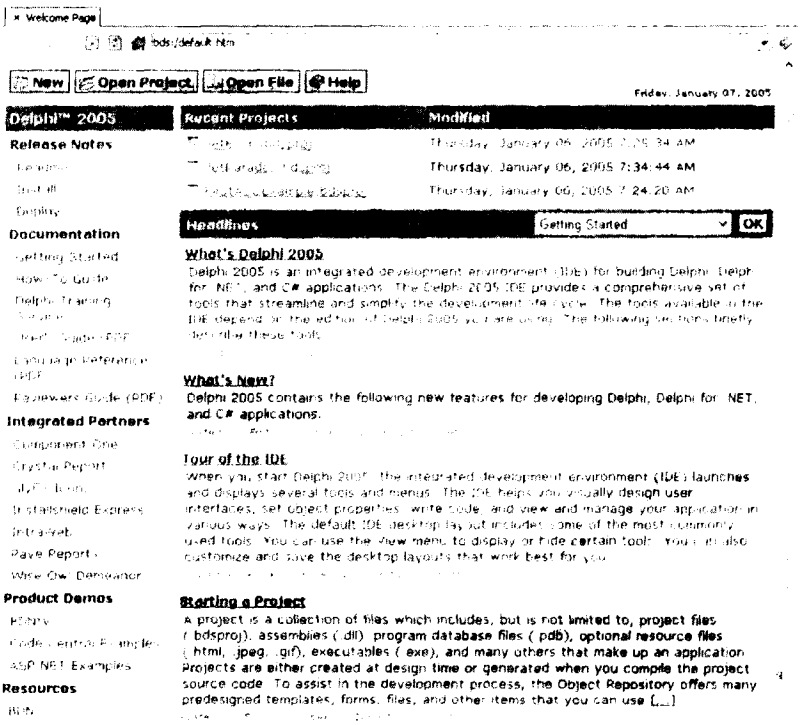


图 1.1　含有 Getting Started 脱机信息的 Delphi 2005 Welcome 页面

在 Headlines 区域内，程序员可以在一组 Borland 相关的 RSS 反馈意见中挑选，这些反馈意见自动更新，并提供与 Delphi 的给定区域相关的最新信息。对 Delphi 程序员来说，最重要的反馈意见是 Borland Developer Network 的 Delphi 部分和 Borland B 日志。当然，这些反馈意见是完全可定制的， Welcome 页面及其菜单也是如此。然而，在给读者提供一组定制技巧以前，笔者首先要说明的是，笔者更喜欢使用一个成熟的浏览器（而且不一定是 Internet Explorer ）来浏览 Web，以及使用一个 RSS 特有的程序保持与 B 日志的联络。

在任何情况下，如果需要定制这个页面，程序员都可以修改它的一些配置文件，它的配置文件全部驻留在 BDS \ 3.0 \ Welcomepage 文件夹中。

- 总体页面结构的定义在 default.htm 文件中。
- xml/menuBar.xml 文件定义页面左边的那些链接，读者可以轻易地提供自己的条目。
- xml/defaulProviders.xml 文件定义 RSS 反馈意见列表。需要注意的是，该文件中有一些已被注释掉的条目，比如一个 Borland B 记录器列表，这些条目随时可以变得可用，只需删去那些注释即可获得更多的 RSS 反馈意见。
- css 文件夹含有程序员可以修改以便定制页面的字体和颜色的 Cascading Style Sheet 文件。
- 尤其是，要想修改 RSS 条目的数量，需要打开 rss.js 文件并修改下面这两行中的第二行（应该是第 249 行）：

```
// limit to 10 entries displayed on welcome page
if (maxItems > 10)
```

Delphi 2005 IDE 结构

Welcome页面被安置在 Delphi 2005 的主区域内，这个区域也由编辑器和设计器使用。设计器允许程序员使用可视化级别的构件（比如当在窗体上放置一个按钮的时候）或非可视化级别的构件（比如当在数据模块上放置一个 DataSet 构件的时候）。图 1.2 中显示了一个嵌入的 Win32 VCL。

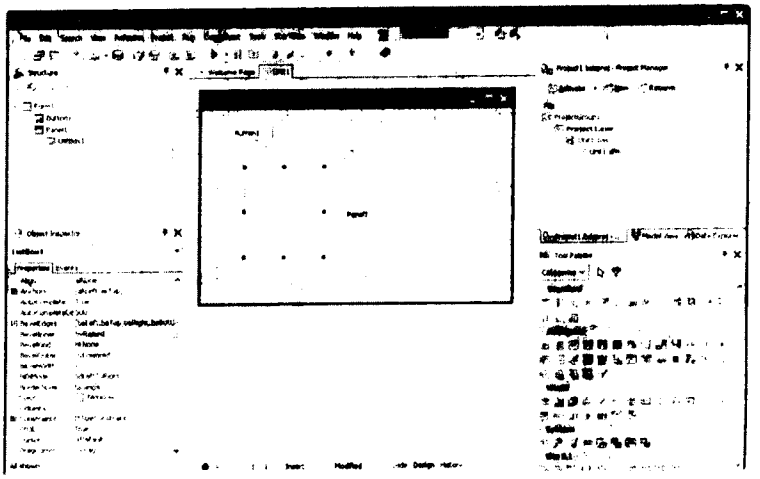


图 1.2 带有一个嵌入式 VCL Form Designer的 Delphi 2005 IDE

在默认情况下，设计器被嵌入到主窗格中；利用页面底部的选卡（而顶部的选卡用于从一个源文件转移到另一个源文件），程序员可以从设计器切换到对应的源代码编辑器。如果愿意，程序员可以像在传统 Delphi IDE 中那样将 VCL 设计器显示在一个单独的浮动窗口中，具体方法是通过禁用可从 Tools【工具】菜单上获得的 Options【选项】对话框的 Environment Options【环境选项】/Delphi Options【Delphi 选项】/VCL Designer【VCL 设计器】部分内的 Embedded【嵌入】设计器复选框。需要注意的是，这个设置仅在重启 IDE 之后才能生效。

IDE 的这个相同窗口也包含代码编辑器，这也是程序员编写 Delphi（或 C#）代码的地方。在可视化环境中，编写代码最显而易见的方法包括响应事件，首先从附加到由程序用户所执行的操作上的事件开始，比如单击按钮或者从列表框中选择一个表项。程序员可以使用相同的方式来处理内部事件，比如涉及到来自操作系统的通知或数据库修改的事件。

大多数 Delphi 程序员喜欢从编写主事件处理代码开始。随着经验越来越丰富，他们逐渐开始编写自己的类和构件，而最后的结果常常是把他们的大部分时间花费在编辑器中，而不是花费在可视化设计器中。由于本书将更多地讨论可视化编程和设法帮助读者掌握 Delphi 的整个威力，所以在接下来的篇幅中，读者见到最多的将是代码，而不是窗体。

再回到 Delphi 2005 IDE 结构的话题。请注意，在主窗格的左右两边（请参见图 1.2），还有其他几个包含各种窗格的窗口。在默认情况下，左边是 Structure View【结构视图】和 Object Inspector【对象检查器】右边有包含 Project Manager【项目管理器】、Data Explorer【数据资源管理器】和 Model View Designer【模型视图设计器】的选卡式单独窗口；而

右底部区域由 Tool Palette 【工具调板】占据。IDE 底部的另一个窗口一般包含编译器结果、搜索结果和一个再加工视图。这个窗口只在必要时才显示出来。另外还有 IDE 所使用的数十个其他窗口，其中包括许多调试器视图。

如果使用过 Delphi 7 或以前的版本，读者将会注意到，尽管布局发生了很大变化，但这些窗格的内容看起来是熟悉的。例如，Structure View 类似于 Delphi 7 的 Object Tree View【对象树视图】，而 Tool Palette 取代了 Components Palette【构件调板】。同时，这两个窗口还可以包含其他信息，比如 Structure View 情况下的代码结构和 Tool Palette 情况下的代码段。在 BDS IDE 中，大多数窗格都是比较灵活的，并且根据用户正在从事什么工作，尤其是用户正在哪个主窗口（可视化设计器、编辑器、UML 查看器等）上进行工作，而包含不同的内容。

当然，只要愿意，程序员可以按照他们的意愿重新排列一切，因而可以四处拖动窗口和窗格，让它们保持浮动状态或者悬挂到 IDE 的某一边，甚至通过在上面移动鼠标，使用锁钉按钮将它们放入视图中。需要注意的是， Options 对话框（如图 1.3 所示）的 Environment Options 主页面含有一个复选框，如果这个复选框被禁用，它将阻止 Delphi 窗口自动停靠在一起。（之所以在此提及这个设置，是因为笔者总是改变它的默认值，改变其默认值的另一个设置是同一个页面上的 Show Compiler Progress【显示编译器进度】复选框。）

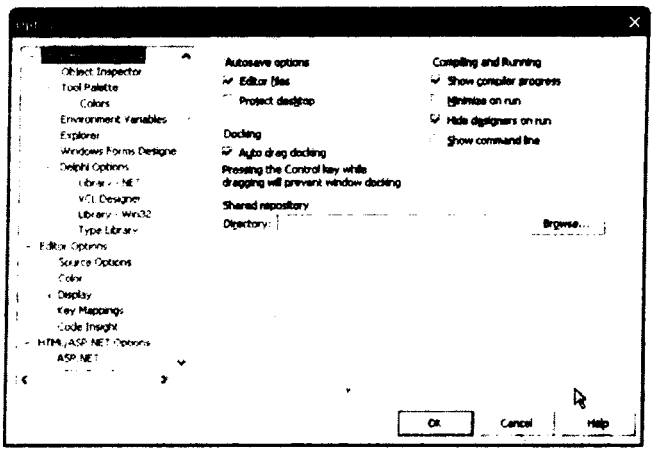


图 1.3 Options 对话框的 Environment Options 主页面

在找到所喜欢的设置之后，程序员可以使用按钮和 Desktop【桌面】工具栏的组合框保存和恢复这个设置（请参见下一节）。

桌面设置

程序员可以用不同的方法定制 Delphi IDE。一典型的方法是打开多个窗口，排列它们，并将它们停靠在一起。然而，我们经常需要在设计的时候打开一组窗口，而在调试的时候打开另一组窗口。与此相似的是，我们在处理窗体的时候需要一种布局，而在仅使用编辑器编写构件或低级代码的时候却需要另外一种完全不同的布局。为这些不同的需求而重新排列 IDE 是一项很繁琐的工作。

因此，Delphi 允许程序员使用一个名称保存 IDE 窗口的一种给定排列并轻松地恢复那些窗口。程序员也可以将这些配置之一变成他们的默认调试设置，以便它在程序员启动调试器时可以自动得到恢复。所有这些特性都可以在 Desktops 工具栏和 View【视图】➤ Desk-

tops【桌面】菜单中访问到。

桌面设置信息保存在 DST 文件中（位于 Delphi 的 bin 目录下），DST 文件实际上是 INI 文件。如果打开这些 DST 文件之一，将会发现它们是非常容易理解的。

菜 单

尽管程序员可能会使用快捷键、快捷菜单和工具栏按钮来完成大部分的任务，但 Delphi 的主菜单栏仍然是程序员与 IDE 进行交互的一种重要方式。菜单栏在响应程序员的当前操作时不会发生太大的变化：程序员需要单击鼠标右键来获得一个完整的操作列表，其中包含程序员能在当前窗口或构件上执行的所有操作。

在 Delphi 的最新版本中，新增的一个重要菜单是 IDE 中的 Window 【窗口】菜单。这个菜单列出打开的浮动窗口和对话框。Window 菜单的确是非常方便的，因为窗口最终常常隐藏在其他窗口的后面，并且寻找起来会很困难。程序员可以使用 Windows 注册表中的一个设置来控制这个菜单的字母表式排列顺序：找到 Sort Window Menu 字符串键值（位于 HKEY_CURRENT_USER\Software\Borland\BDS\3.0 下面）。这个注册表键值使用一个字符串（替代布尔值），其中“-1”和“True”表示真，而“0”和“False”表示假。

To-Do 列表

Delphi 已经拥有相当长一段时间但现在可能仍在滥用的一个特性：To-Do 列表。这是程序员完成项目时必须完成的一组任务——它是一个针对程序员（或程序员们；这个工具在团队开发中是非常方便的）的注释集。尽管这个概念不是什么新东西，但 To-Do 列表在 Delphi 中的关键作用在于它工作起来像是一个两用工具。

通过给一个项目的任一文件的源代码添加特定的 TODO 注释，程序员可以添加或修改 To-Do 表项；然后，程序员将会在这个列表中看到这些对应的表项。另外，程序员还可以可视化地编辑该列表中的表项来修改对应的源代码注释。例如，下面是一个 To-Do 表项在源代码中看起来可能具有的样子：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // TODO -oMarco: Add creation code
end;
```

所有这些特殊注释都显示在 To-Do List 窗口中，而且程序员也可以利用这个窗口的 Edit To-Do Item 对话框编辑这些注释。图 1.4 显示了一个代码段、一个浮动的 To-Do List（为了方便起见，它被默认地挂接在 IDE 的底部）以及位于编辑器窗口中的同一个代码段。

这条两用规则也有一个例外，就是在定义项目级的 To-Do 表项时，程序员必须将这些表项直接添加到 To-Do 列表上。为此，程序员可以在 To-Do 列表窗口中使用 Ctrl+A 组合键，也可以在这个窗口中右键单击并从快捷菜单中选择 Add【添加】命令。这些表项存储在一个特殊的文件中，这个文件的根名称与项目文件相同，但它的扩展名为 .TODO。要想了解这方面的示例，请参见 ToDoTest 演示，其中含有前面所提到的同一个代码段。

一个 TODO 注释可以带有多个选项。程序员可以用 -o 选项（正如前面的代码摘录中所显示的）表示所有者（即输入该注释的程序员），用 -c 选项表示一个类别，或者简单地使用 1 到 5 之间的一个数字表示优先级（0 或没有数字表示没有设置优先级）。例如，使用编辑器

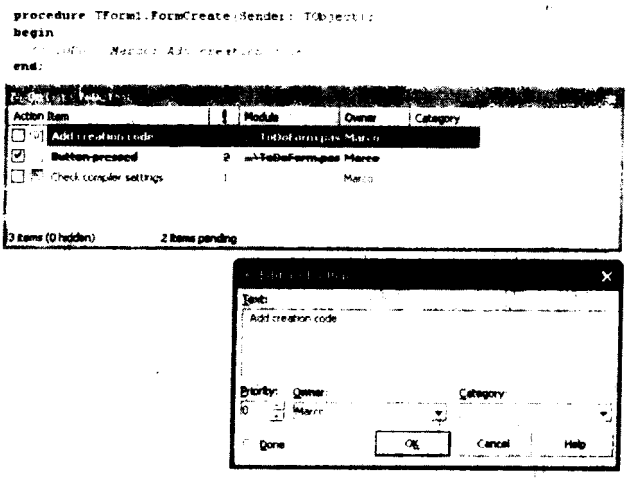


图 1.4　带有一些源代码和对应编辑器的 Edit To-Do Item 窗口

快捷菜单中的 Add To-Do Item 命令（或 Ctrl+Shift+T 快捷键）生成下面这条注释：

```
{ TODO 2 -oMarco : Button pressed }
```

Delphi将会根据注释类型，把冒号之后直到行尾或结束大括号之间的所有字符都看做是该 To-Do 表项的正文。

最后，在 To-Do List窗口中，程序员可以注销一个表项来表明它已被完成。这时，源代码注释将会从 TODO 变为 DONE。程序员还可以手工修改源代码中的注释，并会看到 To-Do List 窗口中出现复选标记。

在这个体系结构中，功能最强大的元素之一是 To-Do List主窗口，它可以在用户键入、分类和过滤源代码文件时从中自动收集 To-Do 信息，并将其作为普通文本或 HTML 表格导出到剪贴板上。

扩展的搜索结果

一般情况下，IDE 底部显示的另一个窗格是 Message 【消息】窗口；该窗口显示编译器消息和搜索结果。即使这个窗口使用起来很简单，但它在 Delphi的早期版本中已经历过几次重大的改进。首先，自 Delphi 7 以后，程序员已经能够在一个不同的选卡中显示搜索结果，以便搜索结果不再像过去那样干扰编译器消息。其次，程序员在每次执行一个不同的搜索时，都可以要求 Delphi在一个不同的页面中显示搜索结果，以便前一个搜索操作的结果仍保持可用。

在 Delphi 2005 中，程序员还可以使用 Find Text【查找文本】对话框中的一个复选框按源代码文件分组搜索结果，这常常会产生一个更易于管理的搜索结果，前提是有比较高的搜索命中率。程序员可以使用 Alt+PageDown 和 Alt+PageUp 组合键来循环遍历这个窗口的各个选卡。(这些命令对其他选卡式视图也同样有效。)

Delphi 编辑器

当第一眼看到 Delphi 2005 时，读者就会意识到 Delphi 编辑器已经发生了重大变化（当

然，除非读者已用过 Delphi 8)。第一个重大变化是编辑器窗口不再是浮动的，尽管程序员可以利用 View【视图】➤ New Edit Window 【新建编辑窗口】命令轻松地打开一个浮动式编辑器。其他重大的变化包括源代码行编号的显示（程序员可以禁用这个特性）、程序员可以用来折叠源代码元素的图标、对各种文件格式的支持等(详细情况,请参见接下来的几节)。

正如我们对带有多种选择的 IDE所要求的，程序员可以使用 Delphi 2005 编辑不同类型的文件，其中包括用 Delphi 和 C# 所编写的源代码文件，还包括用诸如 C++和 VB. NET 之类的其他编程语言所编写的源代码文件和诸如 JavaScript、SQL、XML、HTML、XSL 之类的其他源代码文件。XML和 HTML编辑基于 DTD；这些 DTD 有助于突出语法和自动完成代码。

每个文件上的编辑器设置（包括像 Tab 这样的键盘按键的行为）取决于程序员正在打开的文件的扩展名。程序员可以在 Options 对话框的 Editor Options【编辑器选项】/Source Options 【源代码选项】页面上给每种文件格式都配置这些设置（请参见图 1.5）。这个特性已经有所扩展，并且变得更加开放，以便程序员能够通过为一种基于 XML 的文件格式提供一个 DTD 或者通过编写一个为其他编程语言提供语法突出的自定义向导来配置 Delphi编辑器。Delphi 编辑器的另一个特性（即代码模板）现在是语言所特有的（程序员所预定义的 Delphi 模板在 HTML 或 C# 中没有什么意义）。

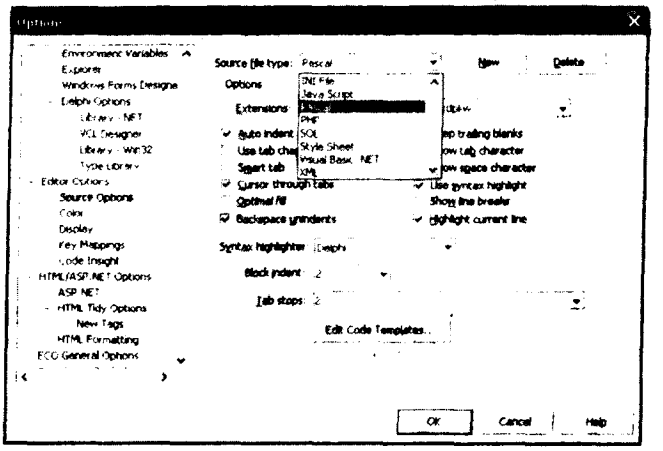


图 1.5 Delphi IDE所支持的多种语言可以与 Options对话框的 Editor Options/Source Options 页面中的各种文件扩展名关联起来

Delphi 编辑器通过使用“选卡式记事本”的概念，允许程序员同时处理多个文件。程序员可以使用 Ctrl+Tab 组合键从编辑器的一个页面跳到另一个页面，也可以使用 Ctrl+Shift+Tab 组合键反方向跳转。利用编辑器上半部分带有单元名称的选卡，程序员可以通过拖动它们来改变其顺序，以便在任一给定时刻仅简单地使用 Ctrl+Tab 组合键就能在正在处理的不同单元之间来回跳转。编辑器的快捷菜单也有一个 Pages【页面】命令，用来列出一个子菜单中的所有可用页面（当装入了很多单元时，这是一个非常方便的特性）。需要注意的是，许多编辑窗口也含有允许程序员在不同视图之间跳转的底部选卡，比如 Code【代码】、Design【设计】和 History【历史记录】。程序员可以利用 Alt+Tab 和 Alt+Shift+Tab 组合键沿着这些视图跳转。

在图 1.5 所示的 Options 对话框中，从 Editor Options 【编辑器选项】区域内可以看到，

还有另外几个对编辑器有影响的选项。但是，程序员必须转到 Environment Options【环境选项】页面才能设置编辑器的 AutoSave【自动保存】特性。这个选项强迫编辑器在程序员每次运行程序时都保存他们的所有源代码文件，以防止程序在调试器中万一出现意外时发生丢失数据的情况。

Delphi 编辑器提供了许多命令，其中包括一些从它的 WordStar 仿真祖先起就已经存在的命令（WordStar 是早期 Turbo Pascal编译器的一部分）。笔者不打算在本书中讨论 Delphi 编辑器的这些设置，因为它们都是非常直观的，并且在联机帮助中均有描述。

> 提示：需要记住的一点是，使用 Cut【剪切】和 Paste【粘贴】命令并不是转移源代码的惟一手段。程序员也可以选择并拖动单词、表达式或整行源代码。另外，还可以通过在拖动的时候按住 Ctrl 键来复制而不是转移文本。

Delphi 2005 代码折叠和区域化

Delphi 允许程序员折叠（或者说压缩）任何声明，其中包括类声明、方法或整段代码（比如一个单元的整个实现部分）。然而，引进这个特性的重要意义和主要理由是程序员能够使用$ REGION 和$ ENDREGION 指令定义可折叠的自定义区域（这两条指令看起来像是编译器指令，但编译器实际上会忽略它们）。

例如，在 RegionsTest 程序中，笔者编写了如下代码：

```
{$REGION 'extra code'}
// this is code you want to hide by default
procedure TestMessage;
begin
  ShowMessage ('Test');
end;
{$ENDREGION}
```

当打开含有这段代码的文件时，将会看到这个代码段已经折叠了起来，并带有一个标题，这个标题与$ REGION 指令中所提供的字符串相对应，如下所示：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  TestMessage;
end;
```

程序员可以使用 BDS/3.0/Editor/Options/Enable Elisions 注册表键值来启用或禁用折叠功能，以及使用 BDS/3.0/Editor/Options/Auto Collapse Region Blocks 注册表键值来确定区域块在打开的文件中的自动压缩。另外，还有一些能够用于区域和折叠的新增快捷键，其中包括：

- 使用 Ctrl+Shift+K+O组合键开关折叠功能（这在恢复折叠功能时要求编辑窗口的手工刷新）；
- 使用 Ctrl+Shift+K+E 和 K+U组合键折叠或展开最邻近的代码块；
- 使用 Ctrl+Shift+K+A 组合键展开所有代码块。

源代码文件编码

过去，Delphi编辑器仅允许程序员处理 ANSI 源代码文件。现在，虽然这仍是默认设置，但 Delphi 代码编辑器内部地处理 UTF-8 编码，并支持以许多种格式保存和转换文件，其中包括 UTF-8和采用了 Little 和 Big Endian 模式的 UTF-16。程序员可以使用编辑器上下文菜单的 File Format 【文件格式】子菜单改变文件格式。

程序员也可以通过添加 BDS \ 3.0 \ Editor \ Options \ DefaultFileFilter 注册表键值来控制默认的文件格式。例如，程序员可以利用字符串 Borland. FileFilter. UTF8ToUTF8 来默认地使用 UTF8 编码文件。

Delphi 2005 代码片断

Delphi 2005 IDE 的另一个新增特性是对代码片断的支持。过去，一直存在定义源代码的可复用块的其他方法（比如程序员仍可以使用 Ctrl+J 组合键显示的代码模板），但是 Tool Palette【工具调板】中所收集的代码片断比较简单，而且使用起来也比较方便。

事实上，使用 Tool Palette ，程序员可以轻松地排列多个类别中的代码片断，将它们从一个类别拖动到另一个类别，也可以通过选择一些源代码并在按住 Alt 键的同时拖动鼠标来创建新的代码片断。要想不必将一个代码片断拖动到代码编辑器就选取它，程序员可以使用 Ctrl+Alt+P组合键选择 Tool Palette，键入用来过滤该片断的词首字母，然后按 Enter键将它粘贴到当前的编辑位置。

代码片断被保存在 BDS \ 3.0 \ Objrepos 文件夹内的 CodeSnippets. xml 文件中，以便它们能够被轻松地从一台计算机转移到另一台计算机，并共享给其他程序员。需要注意的是，还有一个对应于 HTML 编辑器的 HtmlSnippets. xml 文件，它用来保存 HTML 代码片断。

尽管笔者非常喜欢这个特性，但仍希望看到它得到改进。 Tool Palette 中的代码片断最好能够通过打开标题来管理，笔者更喜欢使用不带标题的构件。然而，用于标题的这个设置是整个调板所独有的，因此笔者所希望的东西是不可得的。无论如何，如果考虑让标题在程序员开始过滤调板（通过使用 Ctrl+Alt+P 组合键选择它并开始键入）时显示出来，笔者则更倾向于最终使标题保持在禁用状态。

代码识别

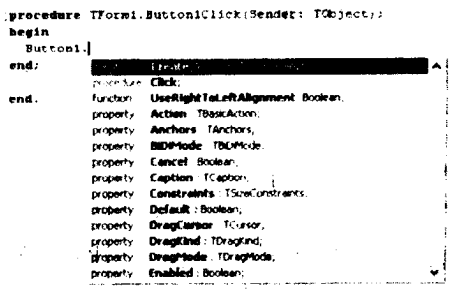
针对目前的几个版本，Delphi 编辑器包含了一个助手集，以便简化代码的编写，这个助手集统称为 Code Insight【代码识别】，并基于程序员所编写的源代码的连续语法分析和程序员所引用的库和系统单元。这就意味着如果程序员的代码含有太多的语法错误，这些特性将无法正常工作。

Delphi 2005 引进了这个家族中的两个新特性，即 Help Insight【帮助识别】和 Error Insight 【错误识别】，它们补充了接下来几节中所讨论的其他 Code Insight【代码识别】特性。在 Options对话框的相应页面中，程序员可以启用、禁用和配置 Code Insight 特性当中的每一个。

注意：Delphi 2005 含有许多 Refactoring【再加工】工具，它们是另外一组基于代码分析的特性。这些工具是十分重要的，因此笔者将在第 10 章中使用一节的篇幅详细讨论它们。

代码生成

代码生成允许程序员通过简单地在对象列表上查找一个对象或通过键入它的首字母来选择这个对象的属性和方法。要想激活对象列表，程序员只需键入一个对象的名称，比如 Button1 ，然后添加句点并等待。要想强行显示这个列表，按 Ctrl+空格键。如果不再需要一个列表，可以通过按 Esc 键删除它。下面是一个示例。



当程序员开始键入时，对象列表根据程序员所插入元素的初始位置过滤它的内容。另外一个特性是，如果函数带有参数，那么生成的代码中将包含圆括号，并且参数列表提示（称为代码参数）被立刻显示出来。代码生成还允许程序员查找赋值语句中的正确值。当程序员在变量或属性的后面键入：=并按 Ctrl+空格键时，Delphi 将会列出具有同一类型的其他所有变量或对象，以及具有同类型属性的所有对象。

当该列表显示出来时，程序员可以右键单击它来改变表项的顺序，进而按作用范围或名称对其分类；程序员还可以改变窗口的大小。即便是经验丰富的程序员，常常也不知道代码生成在类定义中也能工作。在鼠标位于类定义中的时候，如果按 Ctrl+空格键，将会得到一个方法列表，其中包含程序员能够超越的虚方法（包括抽象方法）、该类所实现的各个接口的方法、基类属性以及程序员能够处理的系统消息。简单地选择其中的一个方法（或者一次选择其中的几个方法）将把这个相应的方法（或所有选定方法）添加到类声明上。在这种特定情况下，代码生成列表允许多项选择。

提示：Delphi 2005 能够显示与代码生成列表的当前选定符号相关的 Help Insight【帮助识别】信息。

代码模板

该特性允许程序员插入预定义的代码模板，比如一条内含 begin... end 块的复杂语句。代码模板必须手工激活，具体方法是通过按 Ctrl+J 组合键打开一个含有全部模板的列表。如果在按 Ctrl+J 组合键之前输入几个字母（比如一个关键字）， Delphi 将只列举出以这些字母打头的模板。

程序员可以在 Options 对话框的 Editor Options【编辑器选项】/Source Options【源代码选项】页面中添加定制的代码模板，甚至能够导入或者导出它们（请注意，它们被保存在 DELPHI32.DCI文本文件中）。但是，随着代码片断在 Delphi 2005 中的出现，笔者猜想许多人仍将主动地继续使用代码模板。

代码参数

当键入一个函数或方法时，代码参数将把这个函数或方法的参数的数据类型显示在一个提示或 Tooltip 窗口中。只需输入函数或方法的名称与左括号，参数的名称和类型将会立刻显示在一个弹出的提示窗口中，如下所示：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.ContainsControl(
end;                        Control: TControl
```

要想强行显示代码参数，可以按 Ctrl+Shift+空格键。为了起到进一步提醒的作用，当前参数用黑体显示，而且多个不匹配的重载版本在程序员键入初始参数时将被删去。

Tooltip 表示式估值

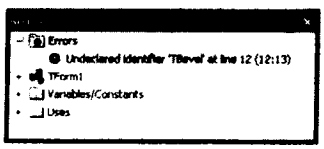
Tooltip 表示式估值是一个调试特性，用来显示鼠标所指向的标识符、属性或表达式的值。对于表达式，程序员通常需要在编辑器中选择它，然后将鼠标移到突出显示的文本上。

错误识别

在 Delphi 2005 中，新增的 Code Insight 【代码识别】特性之一是 Error Insight【错误识别】。这个特性有两种工作方式。第一种方式是当程序员键入 Delphi 无法理解的代码时，Delphi 将用一条红色曲线标出它，类似于字处理器标出误拼单词的方式。当程序员在带有红色曲线的单词上拖动鼠标时，将会看到这个错误的一段简短描述，如下所示：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Buton1
  Undeclared identifier 'Buton1'
end;
```

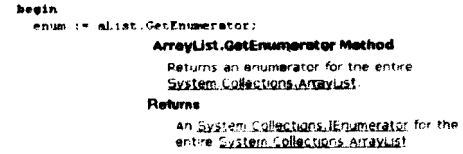
Error Insight 的另一个特性是详细描述在代码编辑期间出现在 Structure View【结构视图】顶部的一个单元的所有错误。有时，程序员只有通过给窗体添加构件才能看到错误出现在 Structure View 中，就像下面这个示例中所显示的那样：



在这种情况下，问题是由这样一个事实引起的：当这个构件被添加时，Delphi 还没有给它的单元添加一个引用。这个操作只有当程序员保存或编译代码时才会发生。由于这个缘故，保存整个单元才是避免这种“伪”错误的最简单方法。

帮助识别

Help Insight【帮助识别】在程序员将鼠标移到一个符号上或者在代码生成窗口中选择一个约束时显示一个摘要帮助窗口。Help Insight 寻找一个含有该符号的相关信息（类型、字段、方法等信息）的 XML 文件。



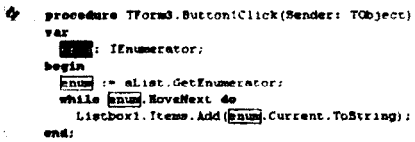
也有用于 Win32 包和 .NET 组件的 XML 文件。如果 Help Insight 没有找到适当的 XML 文件，或者这个文件没有列举出该符号， Help Insight 引擎则显示内部符号信息，比如类型本身、方法参数以及不能真正提供太多附加信息的返回值。

Help Insight 窗口是 Internet Explorer 的一个小型版本。它的输出取决于用来将核心 XML 数据转换成相应 HTML 数据的 XSL 变换文件 HelpInsight. xsl，以及用来确定显示颜色和字体的 CSS 文件 HelpInsight. css。这两个文件均存放在 Delphi 2005 的 Objrepos 文件夹（即 Borland \ BDS \ 3. 0 \ Objrepos 文件夹）中。如果需要定制 Help Insight 的用户界面，程序员可以编辑这个 CSS 文件。

注意 编辑 XSL 文件稍微复杂一些；这个题目将放在第 22 章中加以讨论。

同步编辑

在 Delphi 2005 中，另一个值得一提的新增特性是同步编辑 （Synchronized Editing 或 Sync Edit)。每当选取一段代码时，程序员都会看到一个带有两支铅笔的小图标出现在编辑器的边槽内。通过选择这个图标或者按 Ctrl+Shift+J 组合键，可以激活同步编辑模式，如下所示：



编辑器将利用一个框突出显示重复的单词。程序员可以使用鼠标选择重复的单词之一，或者通过按 Tab 键更轻松选择这个单词；然后，通过键入来开始编辑每次出现的这个单词。

需要注意的是，同步编辑并不是完全基于语法的，而只是一个智能的搜索与替换机制，因此它可能会挑出一个注释内的一个单词（而这个注释实际上可能就是程序员真正需要的，如果它指的是程序员正在修改的一个本地变量）。

编辑器的结构视图

当编辑器处于活动状态时，Structure View【结构视图】窗口将列举出一个单元内所定义的全部类型、变量和例程，以及出现在 uses 语句中的其他单元。正如前面所讨论过的，它可能还列举出单元错误。对于像类这样的复杂类型，结构视图可以列举出详细信息，包括一个含有字段、属性和方法的列表。一旦程序员开始在编辑器中键入时，所有这些信息就会立即被更新。

程序员可以使用 Structure View 在编辑器内做导航。如果双击 Structure View 中的元素之一，编辑器将会跳转到对应的声明。程序员还可以在 Structure View 中直接修改变量、属

性和方法名称。但是，这不像使用 Rename 【重命名】再加工那么灵活。

Structure View 的信息布局是完全可控制的，但控制方式非常不直观。使用 Options 对话框的对应 Environment Options【环境选项】/Explorer【资源管理器】页面，程序员可以配置用于编辑器的 Structure View（请参见图 1.6）。令人奇怪的是，这个页面的名称依然类似于这个特性在 Delphi 7 中所具有的名称：Code Explorer【代码资源管理器】。

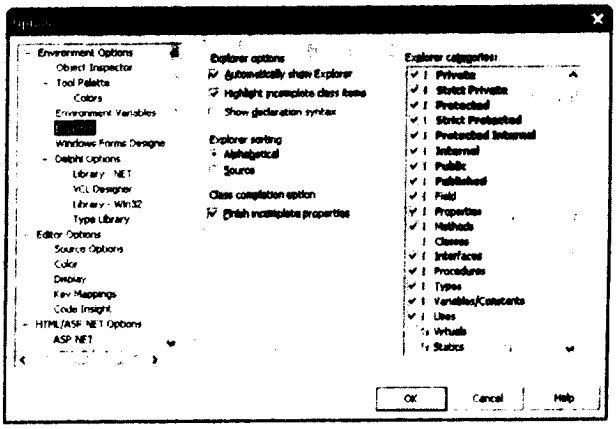


图 1.6 在 Options 对话框的 Explorer 页面中可以配置编辑器的结构视图

当程序员在 Explorer 页面的右边撤消对某一个 Explorer Categories【资源管理器类别】的选择时，Structure View 并不从视图中移走对应的元素——它只是简单地在结构树中添加此节点。例如，如果撤消对 Uses 复选框的选择，Delphi 并不从 Structure View 中将已用单元列表隐藏起来。相反，那些已用单元被列举为主节点，而不是被保存在 Uses 文件夹中。笔者一般禁用 Types、Classes 和 Variables/Constant 选择。

由于 Structure View 树的每个元素都有一个标记其类型的图标，所以按字段和方法的排列方式在重要性上似乎不如按访问说明符的排列方式。笔者的首选是显示单个组中的所有元素，因为这种排列方式只需要最少的鼠标单击就能到达每个元素。

在 Structure View 中选择元素提供了一种导航大型单元源代码的方便手段。当程序员在 Structure View 中双击一个方法时，焦点就会自动跳转到类声明中的对应定义上。

在编辑器中执行浏览

当把鼠标移动到编辑器内的某个符号上时，Help Insight【帮助识别】或 Tooltip Symbol Insight【工具提示符合识别】就会出现在屏幕上，以指出定义该标识符的地方。这个特性可以转变成一个叫做“代码浏览”的导航助手。当按住 Ctrl 键并将鼠标移动到该标识符上面时，Delphi 将创建一个有效的超级链接来指向那个定义。这些链接用蓝色显示并具有下划线，类似于 Web 浏览器中的链接。每当鼠标位于链接上面时，鼠标指针都会变成一只手掌。

例如，Ctrl＋单击 TLabel 标识符可以打开它在 VCL 源代码中的定义。当选择引用时，编辑器将跟踪程序员已经到达过的各个位置，而且程序员可以在这些位置之间来回地跳转——同 Web 浏览器中的情形一样，使用浏览器工具栏上的 Browse Back 和 Browse Forward 按钮，或者使用 Alt＋左箭头键或 Alt＋右箭头键。要想更充分地控制前后跳转，也可以单击 Browse Back 和 Browse Forward 按钮旁边的下拉箭头来检查已经到达过的所有源代码行