Borland C#Builder: 吹响.NET进军企业级市场的号角

左轻侯
2003-5-14

问题：为什么是C#Builder？

Borland公司的三大开发工具Delphi、C++ Builder和JBuilder，长期以来在开发者社群中享有盛名。经过不断的改进，这三大开发工具都已经达到了相当成熟和完善的地步，目前最新的版本号分别是Delphi 7、C++ Builder 6和JBuilder 9。就在这个时候，Borland公司又推出了一个全新的面向Microsoft.NET平台的开发工具——Borland C# Builder（开发代号SideWinder）。这是很长一段时间以来，Borland公司第一次发布全新的开发工具，也是Borland公司第一个针对.NET平台的开发工具（Delphi 7中for .NET的preview版不计），同时还是第一个由获得.NET platform授权的第三方厂商（目前Borland是唯一的一家）发布的.NET开发工具。

同时，C# Builder可能也是Borland有史以来最有争议的开发工具。在C# Builder还没有正式发布之前，在网上各个论坛中就已经充斥着对它的各种推测和质疑。这些议论集中在如下几个问题上：

- .NET是Microsoft独立开发并牢牢控制的平台，C#语言也是由Microsoft一手创建，Borland推出基于.NET与C#的开发工具，是否明智？是否有跟风的嫌疑？
- Visual Studio .NET作为Microsoft自己推出的功能强大的开发工具，在.NET平台上自然具有得天独厚的优势，在Visual Studio .NET的压力下，C# Builder还有生存的空间吗？
- 在.NET平台上，类似于Java，应用程序被编译成中间语言（IL），然后在一个虚拟机上运行。这样，Borland过去引以为自豪的编译器技术实际上已经无用武之地。而且，作为一个完全面向对象的framework，.NET提供了数量丰富的标准控件库，从传统界面的Windows Forms到可视化Web开发的web forms，无所不有。那么，Borland在VCL上的优势也被抵消了。还有什么理由选择C# Builder呢？

在听到Borland公司在开发C# Builder的消息以后，本文作者也和大家一起，充满了这些疑问。但是当接触了Borland公司关于C# Builder的一些资料，特别是实际使用了C# Builder的测试版以后，作者已经在自己心目中对这些问题做了回答。

答案就是：C# Builder并不是Borland公司仓促推出的跟风之作，也不是为了将产品线延展到.NET平台而对Delphi或C++ Builder的照搬。正好相反，C# Builder是Borland寄予厚望的产品，是整合Borland ALM的宏伟战略的下一代开发工具的第一次尝试。C# Builder与Visual Studio.NET之间，与其说是竞争关系，不如说是互补关系。Visual Studio .NET虽然是一套优秀的开发工具，但是C# Builder才是真正为企业级开发量身定造的解决方案。从这个意义上来说，C# Builder正是.NET的一支助推火箭，大大地推动了Microsoft的战略目标——千方百计将.NET平台打进企业级应用的市场。

如果说，这个答案多少显得有些令人难以置信的话，在下文中作者将对C# Builder的几个重要特征进行较为深入的讨论，来为这个回答提出论据。

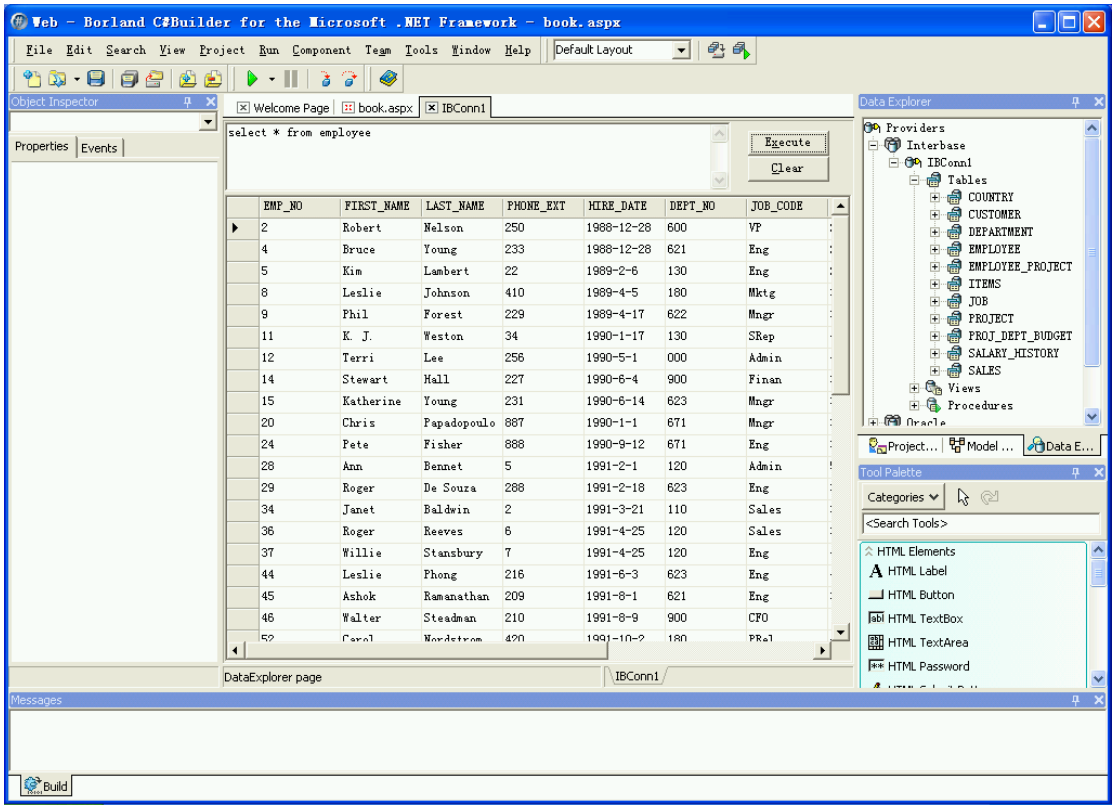
Borland的数据控件：Borland Data Providers

Borland Delphi 中强大的数据库控件系列，给所有使用过的人都留下了深刻印象。初学者惊喜于不需要输入一行代码，只需简单地拖放几个控件就能开发出数据库应用程序的能力，经验丰富的开发者则满足于精巧灵活的架构，可以轻松地开发出高效而结构良好的程序。因此，C# Builder 在这方面的表现，理所当然地成为了备受瞩目的焦点。

在 C# Builder 中，数据库相关的控件由三个部分组成：第一部分是.NET framework 中标准的控件，包括 SqlDataAdapter、SqlConnection、SqlCommand、DataView、DataSet、DataGrid 等等。第二部分是第三方的 ComponentOne 公司提供的 ComponentOne Studio for .NET，包括从 Label、TextBox、Chart 到 ExpressConnection、ExpressTable、ExpressView 等一系列功能强大的控件（在开发工具中捆绑优秀的第三方控件，是 Borland 的一贯策略，也是非常受开发者欢迎的举动）。但是最大的亮点却在于第三部分，即 Borland 提供的 Borland Data Providers for .NET (BDP.NET)。

BDP 只有四个控件，在 Tool Palette 上只占有短短的一行，分别是 BdpConnection、BdpCommand、BdpDataAdapter、BdpCommandBuilder。但是在.NET 对于数据库的支持能力的方面，BDP 带来了质的提升。

首先，BDP 支持多种流行的数据库系统。在.NET framework 1.0 中，只提供了对 MS SQL Server 和 Access 的支持，在 1.1 中也仅仅增加了对 Oracle 的支持。很显然，在复杂的企业级应用中，这种支持是远远不够的。设想一下，如果 JDBC 仅仅能够支持两、三种数据库，怎么能够想象它能在企业级市场上称雄？Microsoft 此举，虽然有助于推动基于 MS SQL Server 的整体解决方案，事实上却构成了让.NET 进入企业级市场的障碍。相比之下，BDP 提供了开放得多的选择，不但内建支持 Interbase、Oracle、DB2、MS SQL Server、Access，而且针对 Informix、Sybase、MySQL、PostGreSQL 等数据库的驱动也将陆续发布。可以说，BDP 提供了对几乎所有的流行数据库的支持，甚至连 MySQL 和 PostGreSQL 这些非主流产品也没有遗漏。



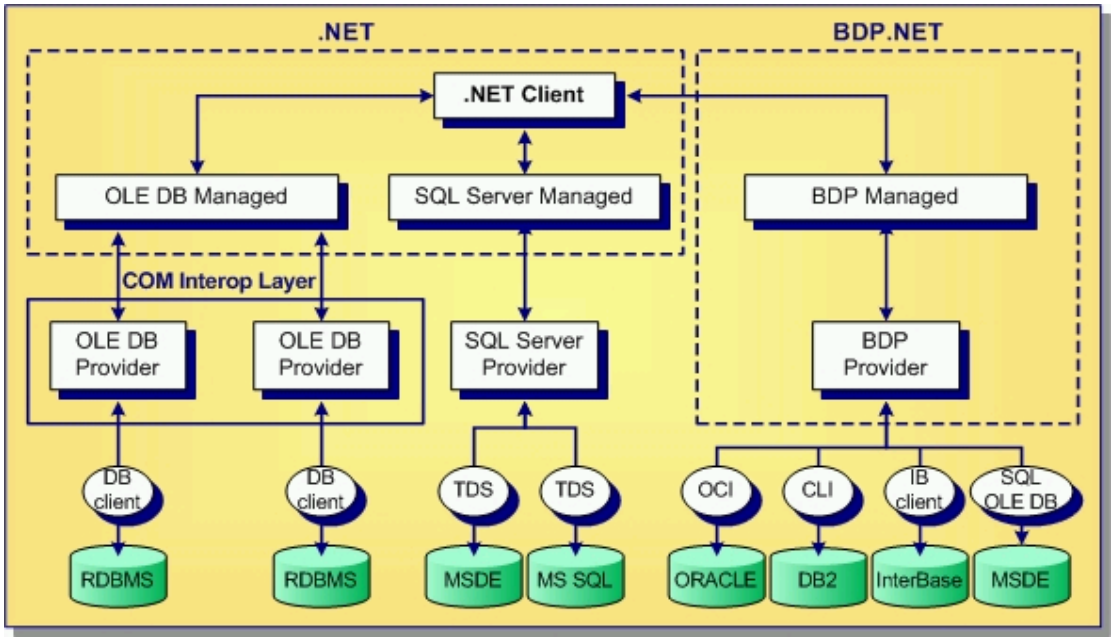
图：集成在 IDE 中的 Data Explorer 和 SQL Window，连接到 InterBase

其次，BDP 支持在设计期查看数据。也就是说，只需要拖放几个控件，设置一下属性，就可以在 C# Builder 的 IDE 中直接看到读取到的数据，包括 BLOB 类型的数据。在 Delphi/C++ Builder 中，这是司空见惯的事情，但是.NET 中可是破天荒第一回。直到 Visual Studio.NET 2003 中，仍然必须通过调用 DataAdapter 的 fill()方法，才能够在运行期看到读取的数据。

第三，BDP 对数据库的访问，比标准的 ADO.NET 要远为迅速和稳定。究其原因，是因为 Borland 公司并不是仅仅简单地在 ADO.NET 规范之下编写新的驱动程序。BDP 中的相关控件，实现了 ADO.NET 的指定接口，因此可以方便地同.NET 的标准界面控件（例如 DataGrid）协同工作。但是，BDP 中的数据读取方式，乃至异常处理机制、数据类型映射等等功能，是由 Borland 完全另起炉灶重新开发的。

在 C# Builder 中，增加了一个 Borland.Data 的名字空间，它包括三个部分：

Borland.Data.Common	包含所有的 Borland Data Provider 的公共类，包括错误与异常类，数据类型枚举，可使用的选项，以及创建自己的 Command, Connection, 与 Cursor 类所需要的接口。
Borland.Data.Provider	包含关键的 BDP.NET 类，诸如 BdpCommand , BdpConnection , BdpDataAdapter，以及其它与 Oracle、DB2、Interbase、MS SQL Server 这样的外部数据源进行通信的类。
Borland.Data.Schema	包含为了创建自己的数据库模型操纵类所需要的接口，以及用于定义元数据的一系列类型与枚举。



图：BDP.NET 的架构

从上图中可以看到，BDP.NET 提供了除 ADO.NET 和 OLE DB 以外的另一个更快速的通道，但它又实现了 ADO.NET 的某些接口，以便与.NET 的 Client 端控件兼容。BDP.NET 通过自己的 Provider 直接与各种数据库进行通讯，而这些 Provider 是由 C++编写生成的原生代码。因此，BDP.NET 具有标准 ADO.NET 不可比拟的效率和稳定性，当然还有开放性。

熟悉 Delphi/C++ Builder/Kylix 的读者，可以会觉得 BDP.NET 似曾相识。是的，BDP.NET 事实上就是 dbExpress 在.NET 平台上的移植。dbExpress 是 Borland 在 Delphi/C++ Builder/Kylix 上推出的取代 BDE 的下一代跨平台数据库引擎，当时 Borland 也承诺将会把 dbExpress 移植到.NET 上。在 C# Builder 中，Borland 果然兑现了自己的诺言。

综上所述，C# Builder 通过 BDP.NET 和第三方控件库，大大地改进了.NET 平台对数据库应用开发的支持，特别是增强了在企业级市场的竞争力。C# Builder 能够做到的很多事情，是标准.NET framework 做得不好或者难以做到的，包括对非 Microsoft 家族数据库的良好支持，设计期的数据显示，以及数据存取的效率与稳定性等等。

模型驱动开发：Together for .NET

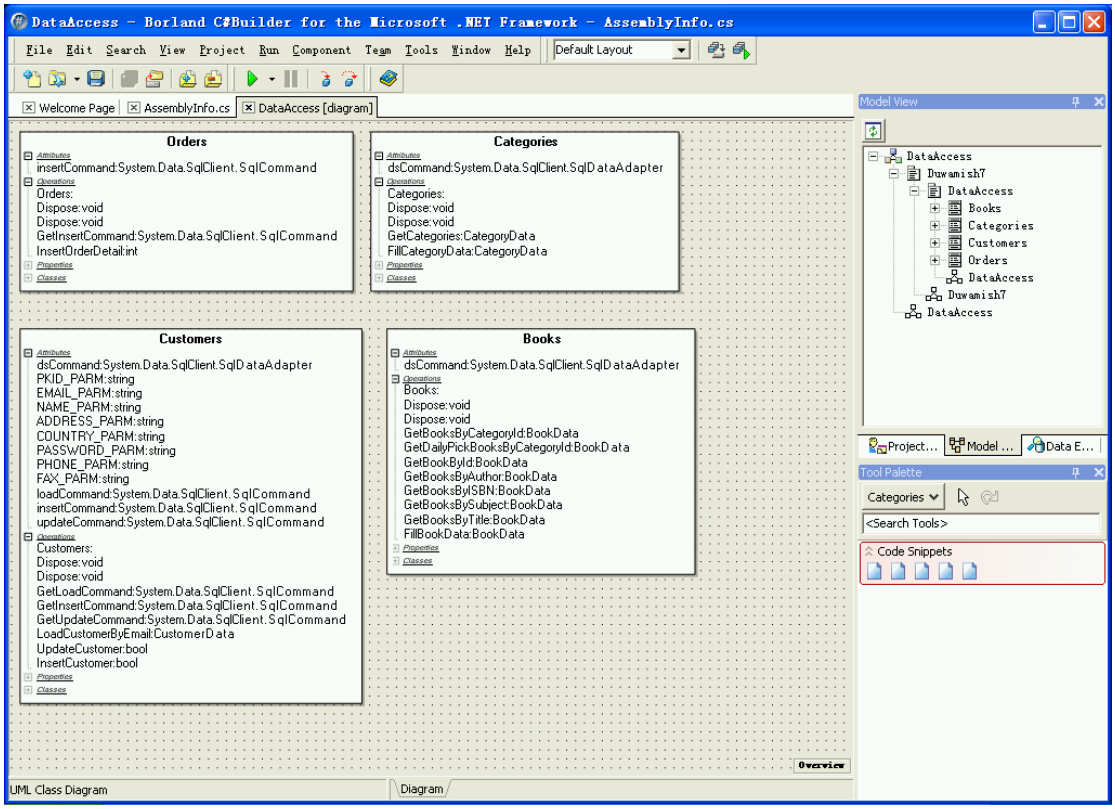
Borland 公司在 2002 年底收购了著名的建模工具 Together，立即引起了业界的震动，因为此举意味着 Borland 公司可能将 Together 强大的 Modeling 功能集成到它的开发工具中。Borland 也果然不负众望，在 2003 年初迅速推出了 Together Edition for JBuilder，为 JBuilder 增加了令人惊叹的 UML Modeling 与 Design Patterns 等功能，使得本来已经名列业界第一的 Java 开发工具 JBuilder 从此更是如虎添翼。（有兴趣的读者，请参阅《程序员》2003 年第 4 期李维先生撰写的《体验 Together Edition for JBuilder》一文。）

Borland 进军.NET 平台的消息传出以后，人们禁不住会猜测，Borland 是否会将 Together 也集成到.NET 平台下的开发工具中呢？.NET 阵营的开发者，是否也可以象 Java 阵营的开发者那样，享有 Together 的强大功能呢？

回答是肯定的。在 C#Builder 1.0 中，已经内建了基本的 Modeling、Reverse Engineering 以及 Design Patterns 的能力，并且 Borland 承诺将在以后的版本中，逐渐将 Together 完整地移植到.NET 下。在此基础之上，C#Builder 将 MDA（Model Driven Architecture，模型驱动架构）概念引入了.NET 平台，这是目前任何其它的.NET 开发工具都不能做到的。MDA 实现了在设计与开发之间的无损转换，从而使开发人员不但能够写出更加严谨、更加高效的代码，而且能够更加方便和直观地设计大型商业系统的对象结构与业务逻辑。也就是说，C#Builder 出现以后，.NET 平台开始拥有与 Java 平台一样强大的建模与代码优化能力，从而在这一方面与 Java 平台站到了同一起跑线上。

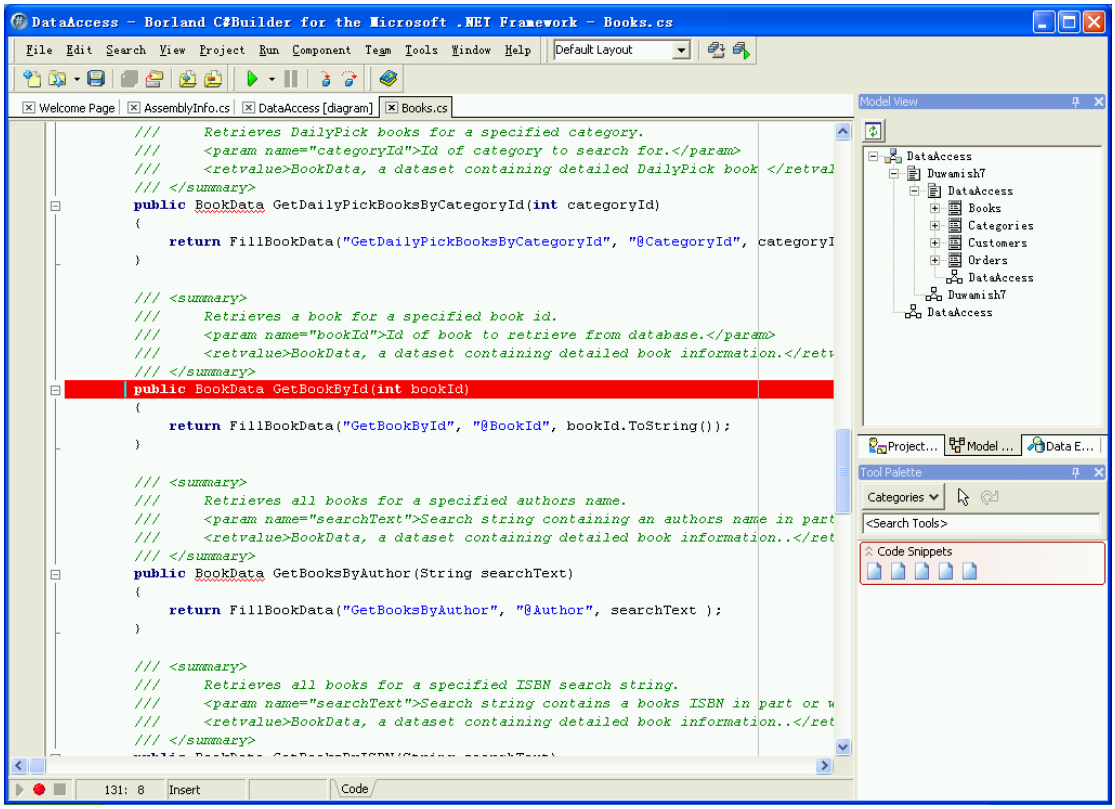
为了测试 C#Builder 在这方面的能力，本文作者使用了一个比较特殊的例子。在学习.NET 过技术的人中，应该有很大一部分不会对 Duwamish 这个单词感到陌生。Duwamish 是 Visual Studio .NET 中附带的一个例子，内容是基于 WEB 方式的多层结构的网上书店，使用 C#编程，后台使用 MS SQL Server 2000。无论是架构设计的严谨，还是代码编写的规范，Duwamish 都堪称经典，因此在许多讨论.NET 开发的技术文章中，都可以看到对 Duwamish 津津乐道的探讨。作者要做的工作，就是用 C#Builder 直接打开 Duwamish 的代码，查看生成的 UML 视图效果。通过这种方式，不但能够考验 C#Builder 的建模能力，而且顺带也考验了 C#Builder 与 Visual Studio .NET 生成的项目以及于 C#语言的兼容性。

在 C#Builder 中，打开 Duwamish 目录下的 DataAccess 子目录，这个目录下的 DataAccess 项目封装了对数据库的直接访问。直接打开 DataAccess.csproj 文件，C#Builde 会提示生成一个新的 DataAccess.bdsproj 文件，然后在右上角的 Project Manager 窗口中就可以看到这个项目的文件列表了。切换到 Model View 窗口，双击 DataAccess，C#Builde 将会自动生成项目的 Code Visualization Diagram，如下图：

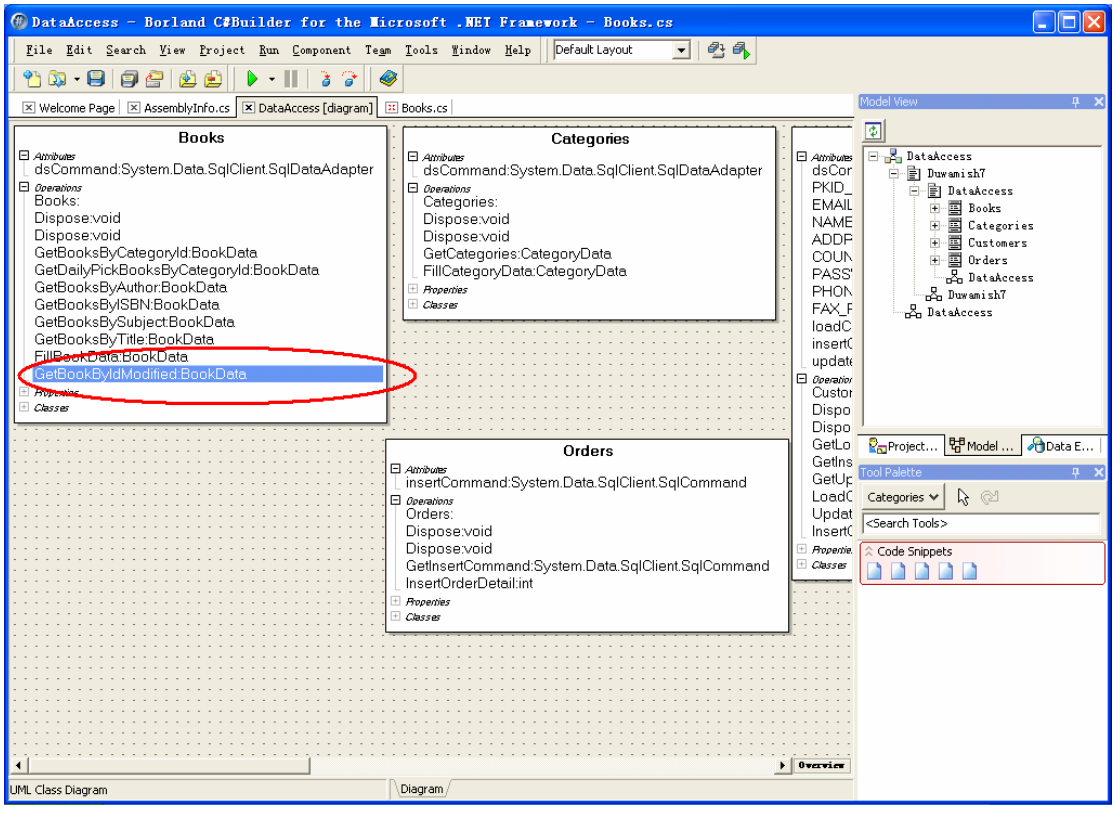


可以看到，DataAccess 中的四个 Class，Books、Categories、Customers、Orders，都已经分别生成了 UML 模型，在每个 Class 下都列出了相应的 Attributes、Operations、Properties 和 Classes（子类）。

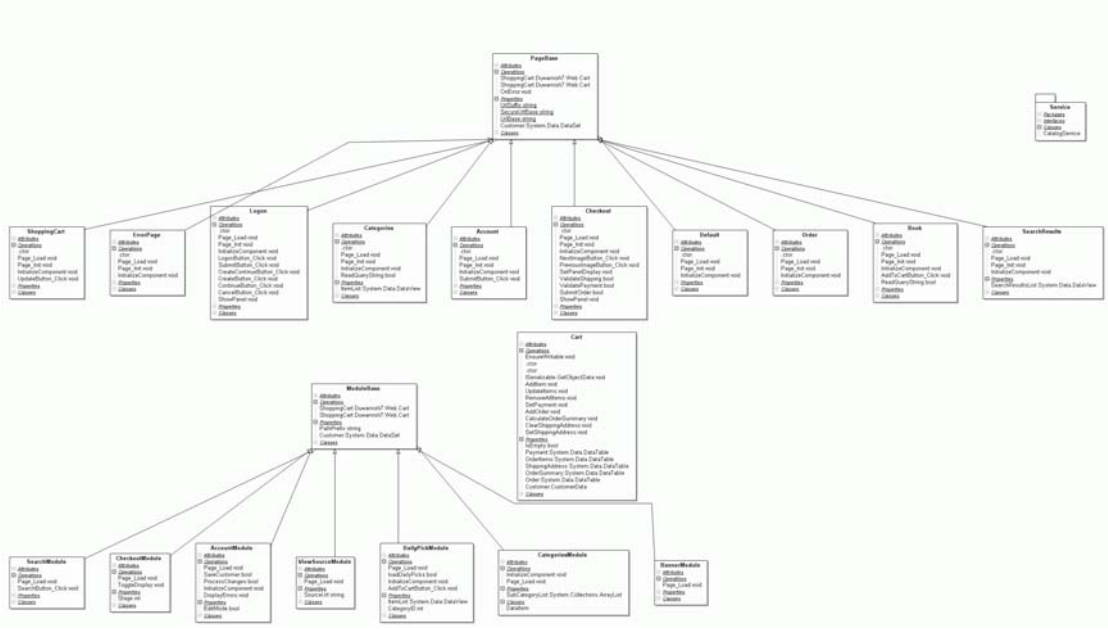
在 Class 的某个方法上右击鼠标，例如 Books 的 GetBookById 方法，在弹出的右键菜单中选中“Open Source”，将会跳转到代码视图中的相应位置，如下图如示：



现在，将 GetBookById 的方法名稍做改动，改为 GetBookByIdModified，再切换到 UML 视图，可以看到在 Books 类中，GetBookById 已经变成了 GetBookByIdModified，如下图所示：



让我们尝试一个更复杂的例子，打开 Duwamish 目录下的 Web 子目录。这是一个包含了大量.aspx 文件以及代码后置（code behind）的.cs 文件的项目。用同样的方法切换到 Code Visualization Diagram，可以得到整个项目的类结构全图，如下图如示。可以看到，所有的类构成了两个继承树，继承关系用箭头的方式表示出来。这样，这个复杂的项目中的逻辑关系就一目了然了。



至此为止，我们已经完整地演示了 C#Builder 的从代码反向生成 UML 视图的能力。自然，C#Builder 也提供了创建和修改 UML 模型、并自动生成相应代码的功能。特别要提到的是，在此基础上，Borland 进一步提出了 ECO（Enterprise Core Objects，企业核心对象）的概念。通过 ECO 技术，在用户设计和创建了应用系统中的对象的 UML 模型以后，C#Builder 能够自动生成相应的 ECO 对象和 Domain 对象。也就是说，用户可以通过类似操纵 Delphi/C++ Builder 组件的拖放方式，方便地建立和设置大型系统中的各种对象。而且，C#Builder 还能够根据 UML 模型自动生成数据库中的原始结构（Database Schema）。此外，C#Builder 提供了将 Design Patterns 应用到代码中的能力：只需指定需要应用的 Patterns 和相关 UML 对象，C#Builder 将自动生成所有相应的对象和代码，如同你在 Together for JBuilder 中看到的那样。

因为作者拿到的这个版本暂时还没有提供 ECO 与 Design Patterns 方面的功能（某些高端的功能只在 Architect 版本中提供），因此无法实际测试 C#Builder 在这些方面的能力。但是，从上面的讨论中不难想象，自从将模型驱动的思想整合到 RAD 开发工具中之后，Borland 已经在逐渐领导下一代软件开发工具和编程方式的革命性变化的潮流。在未来的开发工作中，很可能根据需求直接构建 UML 模型，再从 UML 模型自动生成数据库结构和代码框架。如果说这种方式看起来似乎有点不可思议的话，那么 C#Builder 至少已经部分地达到了目标。这也就是为什么 Borland 将 C#Builder 定位成 Enterprise Solution 的原因——在小型的桌面软件开发中，模型驱动开发可能并不占有明显的优势，但是在涉及到大量对象和复杂逻辑的企业级开发中，这种开发方式将导致生产力的极大提升。

超越.NET：Janeva

在.NET 架构中，通过 Remoting 机制对分布式计算提供了良好的支持，比 DCOM 有了很大的改进。只需要编写很少的代码，.NET 中的对象就可以简单地通过 TCP 或 HTTP 通道进行通信，甚至可以自定义通道。但是，虽然这种分布式模型看上去完美无缺，一旦将它置于真实世界中的大型企业计算环境中时，它的缺点就暴露出来了。原因在于，企业级的分布式组件标准一直是 CORBA 和 J2EE 的天下，除非全部采用 Microsoft 的解决方案，否则就免不了要和它们打交道。而.NET 并没有提供与 CORBA 和 J2EE 进行互操作的令人满意的解决方案。

也许可能已经有人想到了 Web Service。是的，Web Service 是一种简便的、兼容性良好的通讯协议，理论上可以将任何现存的分布式组件结合起来。在.NET 中，也对 Web Service 提供了非常完备的支持。但是，在实际使用中，Web Service 仍然存在着种种难以克服的问题：

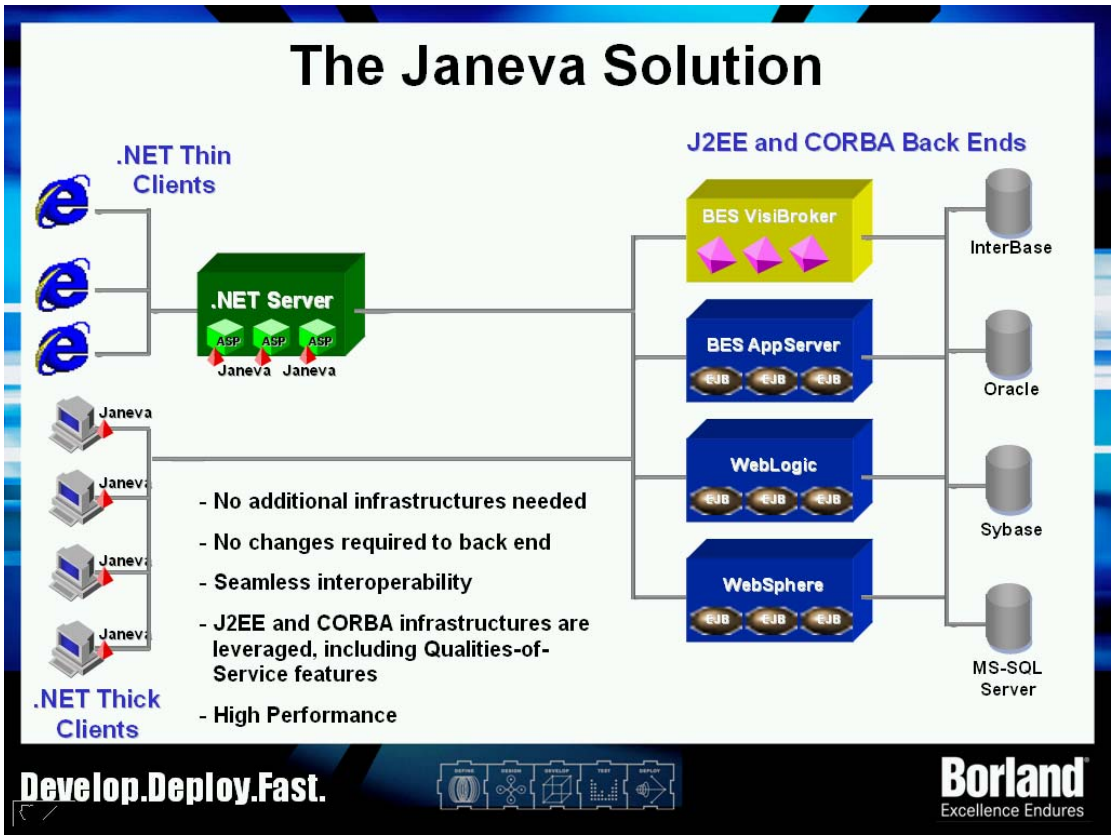
- 首先，它需要付出额外的工作，甚至可能需要对已有的系统进行改动，以让它们支持 Web Service；
- 其次，Web Service 的简单性使它对某些 CORBA/J2EE 的高级特性，例如事务处理、安全性和复杂数据类型，不能够支持或者支持得不好；
- 最后，Web Service 的特性决定了它在效率方面有极大的缺陷，和 Binary 形式的调用相比有数量级上的差距。

这些问题，特别是效率上的问题，使得 Web Service 在可预计的未来仍然无法取得与 CORBA/J2EE 并驾齐驱的地位，因此.NET 也就不能够与 CORBA/J2EE 进行真正有效的通信。这也是.NET 进入企业级市场的主要障碍之一。

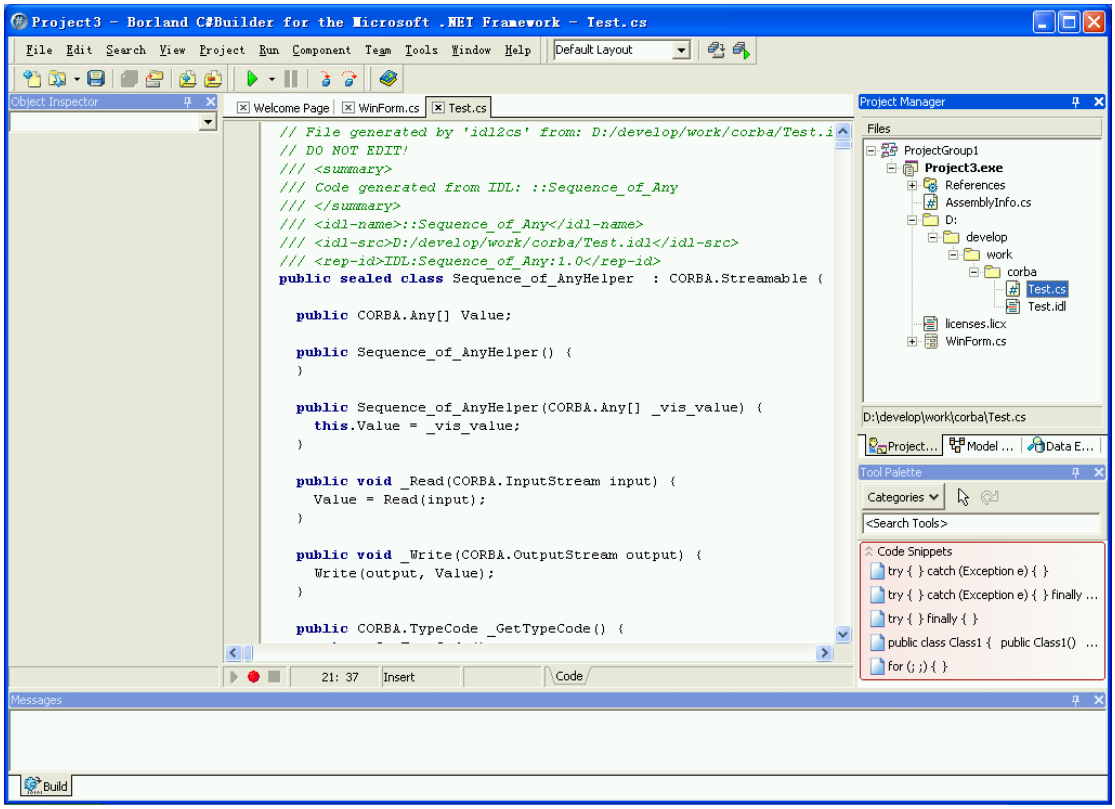
那么，有没有一种方法，可以让.NET/CORBA/J2EE 之间进行无缝的互操作，同时又不致于引起性能和特性上的损失呢？这就是 Borland 公司在 C#Builder 中第一次发布的新技术 Janeva 的任务。

从字面上就可以很明显地看出，Janeva 就是 Java 与.NET 的组合。Janeva 提供这样一种机制，使得运行于.NET 平台下的客户端可以无缝地连接到 CORBA/J2EE 服务端（在以后的版本，还将提供从 Java 客户端连接到.NET 服务端的功能）。Janeva 不但完整地支持 CORBA/J2EE 的各种特性，而且由于采用 Binary 调用的方式，保证了效率（相比 Web Service，有 2-40 倍的效率提升）。

曾经有人质疑过 Janeva 的实现方式，因为直接通过.NET 调用 Java 对象听起来是一件令人难以相信的事情。Janeva 是否是通过 Web Service 模拟出来的，或者是否需要通过 CORBA 服务器进行中转？事实上，Borland 公司的工程师在 Janeva 的底层用.NET 实现了 CORBA/J2EE 的 IIOP 协议，因此 Janeva 的通信方式在效率上和功能上都令人满意，同时无需通过 CORBA 服务端中转就能连接到 Java 对象。这样还带来了一个额外的好处，即原有的服务端完全无须改动以支持新的协议。由于 Janeva 能够自动完成 J2EE 与.NET 之间的数据类型的映射，因此可以支持复杂数据类型。而且，由于 Janeva 的实现遵循了.NET 规范，因此可以支持.NET 下的各种语言，并且在理论上可以用于其它的.NET 开发工具。



通过集成的 Janeva，C#Builder 可以完成许多看上去不可思议的工作。例如，C#Builder 可以在 IDE 中简单地导入 Java 对象的 IDL，然后自动生成相应的.cs 文件，并在此基础上编写 Client 程序，调用远程对象。Delphi 的用户应该对此感到很熟悉，因为这个过程与在 Delphi 中导入 COM 对象自动再生成.pas 文件的过程是非常相似的。不同的是，COM 技术只能跨越 Windows 平台下的语言，Janeva 则跨越了.NET/Java/CORBA，将各大主流计算平台无缝地连接起来了。Borland 公司的工程师演示了一个令人震撼的例子，让 aspx 通过 Janeva 连接到 J2EE 对象，从而使经典的 J2EE 范例 PetStore 在.NET 平台上平稳运行。



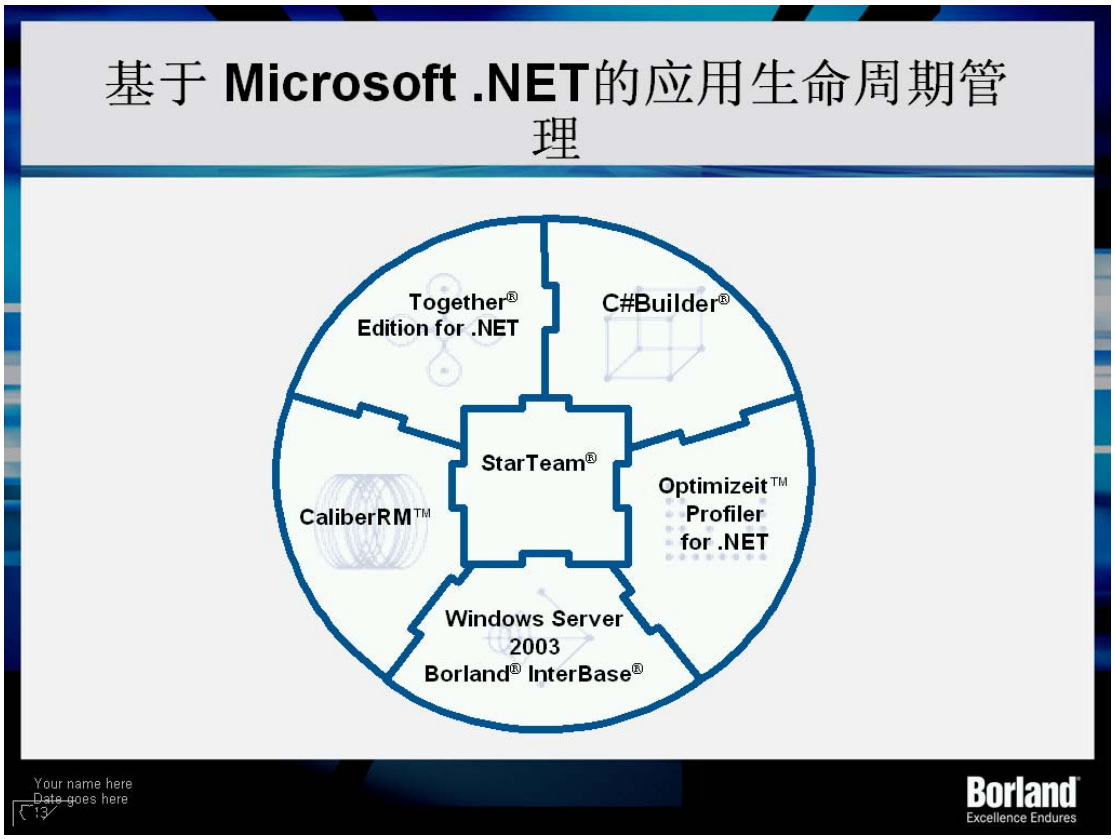
图：通过 Janeva Compile，从.idl 文件自动生成相应的.cs 文件

不难作出结论，Janeva 技术的出现，架起了.NET/Java/CORBA 这几种主流的分布式组件模型之间的桥梁。.NET 作为一个新的竞争者，要在已经相当成熟的企业级市场占有一席之地，这种桥梁对它来说是不可或缺的。C#Builder 集成 Janeva 技术，仅仅这一点，就足以让它成为许多.NET 开发者的首选。

结语：.NET 平台下的企业级解决方案

经过上述这些讨论，相信大家已经对 C#Builder 有了相当的信心。C#Builder 并不是 Delphi/C++ Builder 在.NET 平台上的简单的复制品，而是一个在新的技术和新的开发思想催生下的产物。特别地，C#Builder 对企业级的开发提供了有力的支持，它在这方面的实力是其它.NET 平台下的开发工具难以企及的。反过来说，.NET 在试图进军企业级市场的时候所遇到的许多困难，也因为 C#Builder 的出现而被一一解决。

事实上，Borland 公司并没有将 C#Builder 作为一个单独的工具，而是将它视为 Borland ALM for .NET 战略的一部分。2003 年 4 月，Borland 宣布了 Borland ALM for .NET 战略（另一个是 Borland ALM for Java），如下图。



在此之后，Borland 陆续发布了一系列与此战略相关的软件，包括 Optimizeit Profiler for .NET、C#Builder 以及计划在年底发布的 Together Edition for .NET。很显然，C#Builder 是这个战略中最重要的一环。详细地讨论 Borland ALM for .NET 战略，超出了本文的范围。但是可以想象，随着 Borland ALM for .NET 战略提供的整体解决方案的强力推动，.NET 正在开始成为企业级市场上真正有力的竞争者。