CLR 数据库对象开发指南

CLR 数据库对象开发指南

在 SQL Server 2005 中，最值得一提的开发特性就是与.NET Framework 公共语言运行库（Common Language Runtime——CLR）的整合。CLR 的整合带来了一大堆新的功能，其中包括使用.Net 兼容语言（C#、VB、C++）创建数据库对象。在本次技术手册中，我们将介绍 SQL Server 中的.NET CLR 整合情况，并详细介绍如何创建数据库对象，相信在短时间内大家就可以掌握 SQL Server 的 CLR 开发。

.NET framework CLR 基础

.NET Framework CLR 与 SQL Server 2005 数据库引擎集成的非常紧密。实际上，SQL Server 数据库引擎是以 CLR 为基础宿主的。与同 DB2 和 Oracle 与.NET 的集成相比，这一紧密的集成使得 SQL Server 2005 具备了与.NET 集成的几个明显的优势。

- 开发 CLR 数据库对象：CLR 架构
- 开发 CLR 数据库对象：CLR 程序集

创建 CLR 数据库对象

在 SQL Server 程序集创建以后，你可以使用 SQL Server Management Studio 执行 T-SQL 的 CREATE PROCEDURE，CREATE TRIGGER，CREATE FUNCTION，CREATE TYPE，或者 CREATE AGGREGATE 语句，使用 EXTERNAL NAME 从句指向你之前创建的程序集。

- 创建 CLR 数据库对象
- CLR 存储过程（上）
- CLR 存储过程（下）
- CLR 触发器（上）

- CLR 触发器（下）
- 用户定义类型（上）
- 用户定义类型（下）
- 用户定义函数

调试 CLR 数据库对象

在.NET Framework，Visual Studio 2005 和 SQL Server 2005 的集成功能中，最酷的特性之一就是支持调试你创建的 CLR 数据库对象的能力。如此紧密的整合，使得 SQL Server 在与 Oracle 和 DB2 等同类数据库的竞争中遥遥领先。

- 调试 CLR 数据库对象
- CLR 聚合功能（上）
- CLR 聚合功能（下）

开发 CLR 数据库对象：CLR 架构

.NET Framework CLR 与 SQL Server 2005 数据库引擎集成的非常紧密。实际上，SQL Server 数据库引擎是以 CLR 为基础宿主的。与同 DB2 和 Oracle 与.NET 的集成相比，这一紧密的集成使得 SQL Server 2005 具备了与.NET 集成的几个明显的优势。在下图 3-1 中，你可以看到 SQL Server 2005 数据库引擎和 CLR 集成的概要介绍。

如图 3-1 所示，你可以看到，CLR 是 SQL Server 数据库引擎内的宿主在 SQL Server 数据库引擎内的。SQL Server 数据库使用专用 API 或者宿主层与 CLR 交互，使 Windows 操作系统与 CLR 相连接。CLR 宿主在 SQL Server 数据库内使得 SQL Server 数据库引擎可以控制 CLR 的几个重要方面，其中包括：

内存管理

线程

垃圾回收

DB2 和 Oracle 的实现都使用 CLR 作为外部进程，这意味着 CLR 和数据库引擎都会竞争系统资源。SQL Server 2005 进程内宿主 CLR 给 Oracle 或者 DB2 使用的外部实现提供了几个重要的优势。首先，进程中宿主使得 SQL Server 可以控制 CLR 的执行，使得像内存管理，垃圾回收和线程管理这些重要的功能都在 SQL Server 数据库引擎的控制之下。在外部实现中，CLR 将独立地管理这些功能。数据库引擎对系统需求整体上有一个更好的效果，可以比 CLR 本身能更好地管理内存和线程。最后，进程中宿主 CLR 将提供更好的性能和可扩展性。

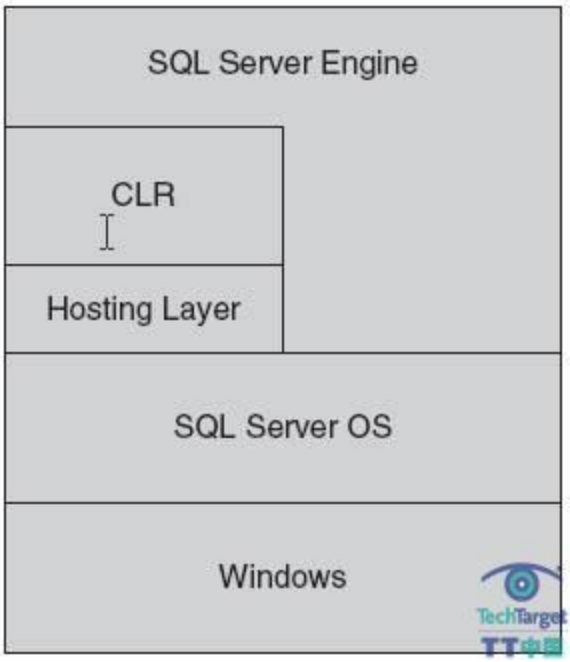

图 3-1: SQL Server CLR 数据库架构

启用 CLR 支持功能

默认状态下，SQL Server 数据库引擎中的 CLR 支持功能是处于关闭状态的。这是为了确保 SQL Server 的更新安装不会在管理员没有明确参与的情况下无意中启用新功能。要启用 SQL Server 的 CLR 支持，你需要使用 SQL Server sp_configure 系统存储过程的高级选项，请看下面的列表：

```
sp_configure 'show advanced options', 1
GO
RECONFIGURE
GO
sp_configure 'clr enabled', 1
GO
RECONFIGURE
GO
```

CLR 数据库对象组件

要创建.NET 数据库对象，你要从编写管理代码开始，可以用任何一种.NET 语言，比如 VB，C#或者 Managed C++，然后把代码编译成.NET 的 DLL(动态链接库)。最常见的实现方式就是使用 Visual Studio 2005 创建一个新的 SQL Server 项目，然后执行编译构建生成 DLL。当然，你也可以使用你自己喜欢的编辑器编写.NET 代码，然后使用.NET Framework SDK 把代码编译成.NET 动态库。ADO.NET 是连接 CLR 动态库和 SQL Server 数据库的中间件。一旦.NET 动态库创建好以后，你需要把该动态库注册到 SQL Server，创建一个新的 SQL Server 数据库对象叫做程序集(assembly)。程序集本质上就是把.NET 动态库封装了一下。然后你可以创建一个新的数据库对象(比如存储过程或者触发器)指向 SQL Server 程序集。在图 3-2 中，你可以看到创建 CLR 数据库对象的大致过程。

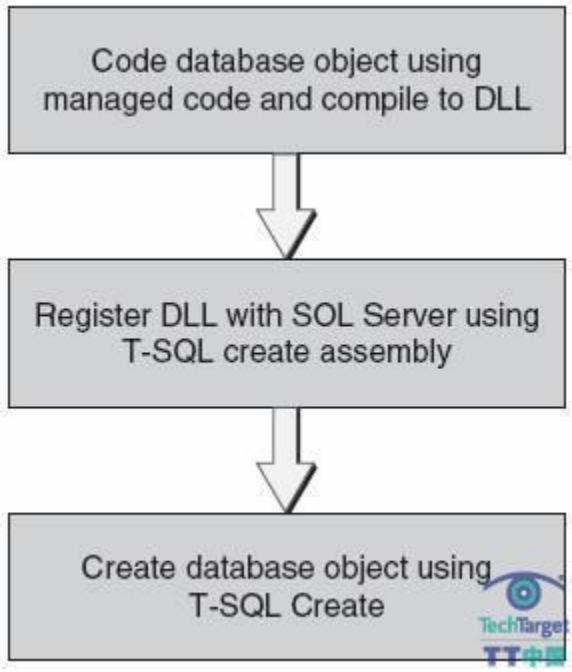

图 3-2: 创建 CLR 数据库对象

(作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国)

开发 CLR 数据库对象：CLR 程序集

SQL Server .NET 数据提供器

如果你熟悉 ADO.NET，你可能想明确地了解一下 CLR 数据库对象怎样连接数据库。毕竟，ADO.NET 使用基于.NET 数据提供器(比如 SQL Server .NET 数据提供器)的客户端建立数据库连接，是用联网库连接的。为了一次数据库调用遍历整个系统的网络支持，对于直接运行在服务器上的代码来说，不是处理这一问题最高效的模式，但对于客户端应用来说是一件好事。为了解决这一问题，微软创建了新的 SQL Server .NET 数据提供器。SQL Server .NET 数据提供器建立了向 SQL Server 数据库的内存内连接。

程序集

在 CLR 对象的编码完成以后，你可以使用该代码创建 SQL Server 程序集。如果你使用的是 Visual Studio 2005，那么你只要简单地选择“部署”选项就行了，它可以负责创建 SQL Server 程序集和创建目标数据库对象的处理。

如果你没有使用 Visual Studio 2005 或者你想手工执行部署步骤，那么你需要复制.NET 动态库到一个指定的位置。然后，你可以使用 SQL Server Management Studio 执行 T-SQL 的 CREATE ASSEMBLY 语句，引用刚才路径的.NET 动态库，如下面列表所示：

```
CREATE ASSEMBLY MyCLRDLL

FROM 'SERVERNAMECodeLibraryMyCLRDLL.dll'
```

CREATE ASSEMBLY 命令有一个参数包含了动态库的路径，从该路径可以把动态库加载到 SQL Server 数据库。这个路径可以是一个本地路径，但是更多时候它可能是一个网络共享文件路径。在执行 CREATE ASSEMBLY 时，动态库被复制到主数据库中。

如果程序集被更新或者变成不推荐使用的了，你可以使用 DROP ASSEMBLY 命令删除程序集。如下所示：

```
DROP ASSEMBLY MyCLRDLL
```

因为程序集存储在数据库中，当程序集的源代码被修改后程序集被重新编译，必须首先使用 DROP ASSEMBLY 命令把程序集从数据库中删除掉，然后在更新从 SQL Server 数据库对象中反射出来之前使用 CREATE ASSEMBLY 命令重新加载程序集。

你可以使用视图 sys.assemblies 查看已经添加到 SQL Server 2005 中的程序集。SQL 如下：

```
SELECT * FROM sys.assemblies
```

既然程序集是使用外部文件创建的，你还可能想看看用来创建那些程序集的文件。你可以使用 sys.assembly_files 视图做到这一点。SQL 如下：

```
SELECT * FROM sys.assembly_files
```

（作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国）

创建 CLR 数据库对象

在 SQL Server 程序集创建以后，你可以使用 SQL Server Management Studio 执行 T-SQL 的 CREATE PROCEDURE，CREATE TRIGGER，CREATE FUNCTION，CREATE TYPE，或者 CREATE AGGREGATE 语句，使用 EXTERNAL NAME 从句指向你之前创建的程序集。

程序集被创建以后，动态库被复制到目标 SQL Server 数据库，程序集也被注册了。下面的代码展示了使用 MyCLRDLL 程序集创建 MyCLRProc 存储过程的方法。

```
CREATE PROCEDURE MyCLRProc

AS EXTERNAL NAME

MyCLRDLL.StoredProcedures.MyCLRProc
```

EXTERNAL NAME 从句是 SQL Server 2005 中的新特性。这里 EXTERNAL NAME 从句表示将要使用 SQL Server 程序集来创建存储过程 MyCLRProc。该动态库被封装在 SQL Server 程序集中，可以包含多个类和方法;EXTERNAL NAME 语句使用如下语法指定从程序集中使用正确的类和方法：

```
Assembly Name.ClassName.MethodName
```

在前面的例子中，注册的程序集名为 MyCLRDLL。在程序集中的类是 StoredProcedures，类中要执行的方法是 MyCLRProc。

关于怎样使用 Visual Studio 2005 实际创建一个新的受管代码项目的具体例子会在下一节中详细阐述。

创建 CLR 数据库对象

前面的部分用一些手工操作 CLR 数据库对象例子的步骤展示了一个概要流程，是为了帮助你理解 CLR 数据库对象的创建和部署过程。然而，虽然也可以手工创建 CLR 数据库对象，但那肯定不是最高效的方法。Visual Studio 2005 专业版，企业版，团队系统版都提

供了帮助创建 CLR 数据库对象，并部署和调试 CLR 数据库对象的工具。在本章的下一部分，你会看到怎样使用 Visual Studio 2005 创建 CLR 数据库的每一种对象。

注意：SQL Server 项目的创建只在 Visual Studio2005 专业版或者更高版本中才支持。在 Visual Studio 标准版或者 Visual Studio 的早期版本中是不支持的。

（作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国）

CLR 存储过程（上）

存储过程是最常见的数据库对象之一，你可以使用某一种.NET 语言创建存储过程。CLR 存储过程的最大价值之一是替换现存的扩展存储过程。T-SQL 只可以访问数据库的资源。为了访问外部系统资源，微软已经在 SQL Server 中提供了支持，叫做扩展存储过程（extended stored procedure）。扩展存储过程是不受管理的动态库，运行在 SQL Server 进程空间，基本上可以做到任何标准可执行程序能做到的事，包括访问数据库外部的系统资源，比如读写文件系统，读写注册表和访问网络。

然而，因为扩展存储过程与 SQL Server 数据库引擎运行在相同的进程空间，所以在扩展存储过程中的程序 bug，内存非法使用和内存泄露都可能潜在地影响 SQL Server 数据库引擎。CLR 存储过程解决了这一问题，因为它们是用受管理的代码实现的，运行在 CLR 的范围之内。CLR 存储过程的另一个优势是取代包含复杂逻辑和包括业务规则的现有 T-SQL 存储过程，这些存储过程难以用 T-SQL 实现。

CLR 存储过程可以利用.NET Framework 类库提供的内建功能，使得添加像复杂数学运算或者数据加密这类功能变得相对比较容易。另外，既然 CLR 存储过程是编译执行的，而不是像 T-SQl 那样是解释执行的，那么他们对于那些执行多次的代码，可以带来巨大的性能优势。然而，CLR 存储过程本身并不是专门为取代 T-SQL 存储过程而设计的。T-SQL 存储过程仍然是数据为中心的存储过程的最佳选择。

要在 Visual Studio 2005 中创建 CLR 存储过程，首先选择“新建|项目”选项，然后选择“SQL Server 项目”模版，如下图 3-3 所示：

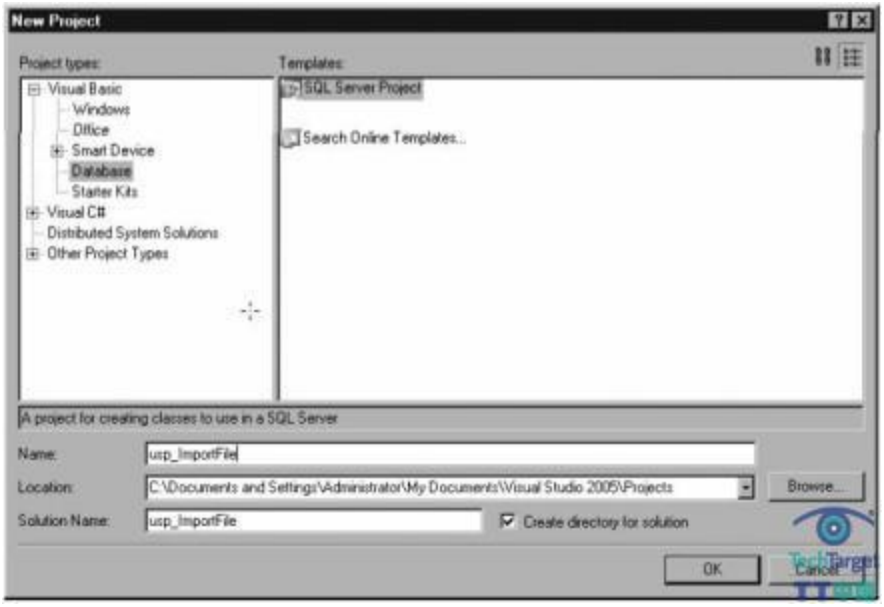

图 3-3：创建一个新的 SQL Server 存储过程项目

给项目指定一个名称，然后点击确定创建项目。在本例中，你可以看到我使用“usp_ImportFile”作为我的存储过程名称。这个存储过程展示了你怎样用 CLR 存储过程取代扩展存储过程。在本例中，CLR 存储过程将读取一个文件的内容，然后把内容存储到 SQL Server 数据库的列中。在给项目命名之后，点击确定。在 Visual Studio 生成项目代码之前，它会显示新建数据库引用对话框，如下图 3-4 所示。

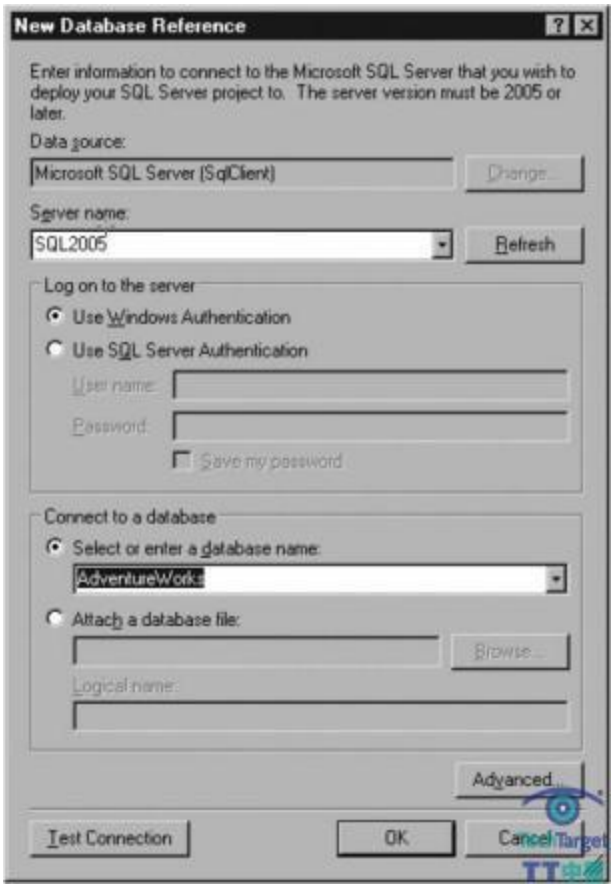

图 3-4：新建数据库引用对话框

Visual Studio 2005 使用“新建数据库引用”对话框创建指向 SQL Server 2005 系统的连接。该连接将用来调试和部署完成后的项目。拉开“服务器名称”下拉列表，选择你想在本项目中使用的 SQL Server 服务器名称。然后选择你想使用的认证类型，选择用来部署 CLR 存储过程的数据库。在图 3-4 中你可以看到我已经选择了名为 SQL2005 的 SQL Server 系统。该项目将使用 Windows 集成身份验证连接，存储过程会被部署到“AdventureWorks”数据库。你可以通过点击测试连接按钮验证连接属性。连接属性按照你的意图设置好了以后，点击确定。所有需要的引用会自动添加到 SQL Server 项目中，Visual Studio 2005 会生成一个 SQL Server 启动项目。

接下来，创建 CLR 存储过程。你可以选择“项目|添加存储过程”选项，显示“Visual Studio 已安装模版”对话框。如下图 3-5 所示。

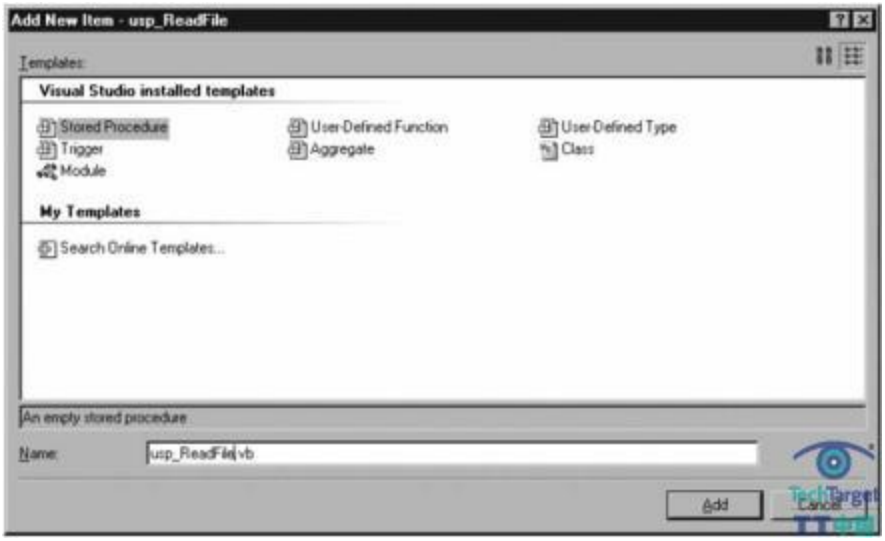

图 3-5：添加一个 CLR 存储过程

在“添加新项目”对话框中，从模版中显示的模版列表里选择“存储过程”选项，然后在对话框底部的名称文本框中指定存储过程的名称。这里你可以看到存储过程将用“usp_ImportFile.vb”源文件创建。Visual Studio 2005 将为给存储过程给你的项目中添加一个新类。生成的类文件在你的存储过程名之后命名，而且所有需要的导入目录和该存储过程的启动代码都会自动添加。你可以看到 SQL Server CLR 存储过程模版，如图 3-6 所示。

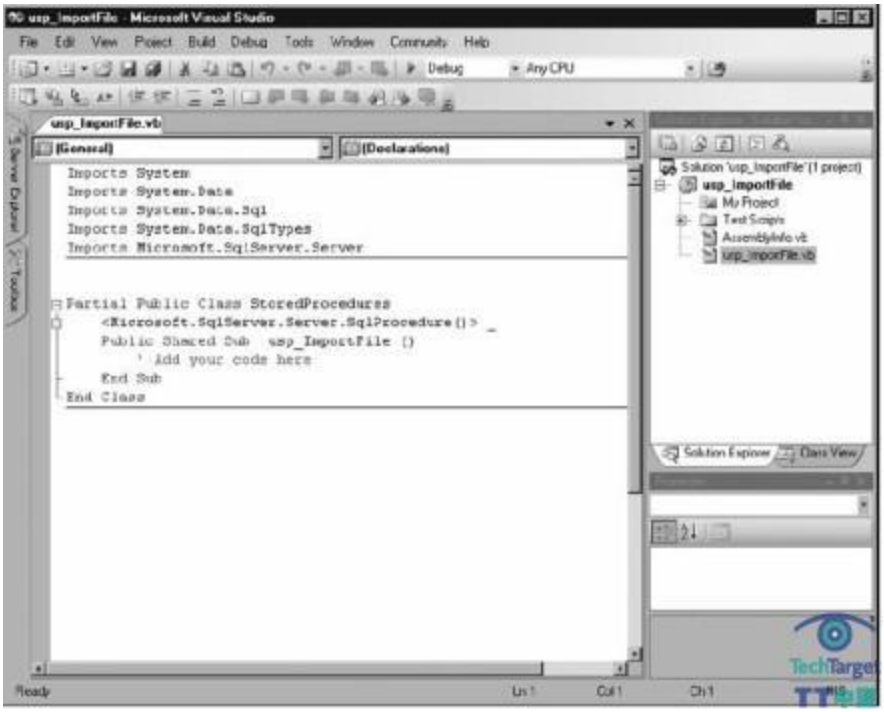

图 3-6：CLR 存储过程模版

默认情况下，SQL Server .NET 数据提供器会自动作为引用添加进来，同时包括 System.Data.SqlServer 命名空间的声明。另外，你还可以看到 System.Data 引用，它提供了对 ADO.NET 的支持和对它面向数据的对象（例如 DataSet）的支持，你还可以看到 System.Data.SqlTypes 命名空间，它提供对 SQL Server 数据类型的支持。

接下来您就可以填写剩下的代码，来使存储过程工作了。下面的例子展示了创建一个简单的 CLR 存储过程需要的源代码，该存储过程的功能是把一个文件中的内容导入到数据库中的一个 varchar 类型或者 text 类型的字段列中：

```
Imports System
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server
Imports System.IO
Partial Public Class StoredProcedures
_
```

```
        Public Shared Sub usp_ImportFile _
    (ByVal sInputFile As String, ByRef sColumn As String)
            Dim sContents As String
                    Try
    Dim stmReader As New StreamReader（sInputFile）
        sContents = stmReader.ReadToEnd（）
            stmReader.Close（）
            sColumn = sContents
            Catch ex As Exception
    Dim sp As SqlPipe = SqlContext.Pipe（）
            sp.Send（ex.Message）
                End Try
                End Sub
                End Class
```

在这段代码中，第一个要注意的要点是导入 Microsoft.SqlServer.Server 命名空间的指令。它使得“usp_ImportFile”项目使用 SQL Server .NET 数据提供器时不需要引用完整的限制名称。第二点要注意的是方法名之前的属性;它告诉编译器该方法将会编译为一个 SQL Server 存储过程。接下来，你可以看到这个存储过程默认的类名被设置为“StoredProcedures”。这个类包含了一个名为“usp_ImportFile”的共享方法，该方法接受两个参数：一个字符串参数，用来指定待导入文件的文件名;第二个输入参数指定用来存储文件内容的数据库字段列名。在 C#语言中，该方法必须定义为静态（static）类型。在 VB.NET 代码中，该方法需要被定义为共享（Shared）的。

在“usp_ImportFile”方法内部，声明了一个名为“sContents”的字符串对象，它用来存储文件的内容。接下来，Try-Catch 代码段用来捕获在文件导入过程中可能出现的任何异常。在 Try-Catch 代码段内，创建了一个新的名为“stmReader”的 StreamReader ，用来从操作系统中读取文件。待读取文件的文件名参数会被传递到 StreamReader 的实例调用中。然后使用“stmReader”的 “ReadToEnd”方法读取文件的整个内容到字符串变量“sContent”中。在文件内容被读取之后，流读取变量“stmReader”被关闭，“sContents”变量的内容会被赋到 SQL Server 的字段列中。

如果在输入文件读取过程中出现了任何错误，Try-Catch 结构中的 Catch 代码段会被执行。在 Catch 代码段中，创建有一个名为“sp”的 SqlPipe 对象，它用来把错误信息送回到存储过程的调用者那里。这段代码使用了 SqlPipe 对象，它是在 CLR 和调用代码之间传递信息的管道。这里，SqlPipe 对象使存储过程可以传递信息给外部调用者。

(作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国)

CLR 存储过程（下）

设置存储过程的安全性

到这个时候，存储过程的代码就完成了，但是由于安全性问题，它不能执行。默认情况下，SQL Server CLR 对象只能访问数据库资源，它们不能访问外部资源。在本例“usp_ImportFile”中，存储过程需要访问文件系统，所以默认安全设置需要修改一下。要启用对外部资源的访问，你需要打开项目属性，点击数据库 Tab 页。然后在许可级别下拉列表中，你可以把它的值从安全(Safe)修改为外部(External)。更多关于 CLR 安全选项的信息将会在本章的后续部分加以表述。

部署存储过程

在 CLR 存储过程源代码编译成程序集以后，你可以添加程序集到数据库，创建 CLR 存储过程。你可以以两种方式做到这一点。如果你使用 Visual Studio 2005 创建 SQL Server CLR 数据库对象，那么你可以交互式地直接从 Visual Studio 中部署 CLR 存储过程。要把存储过程部署到 SQL Server，在 Visual Studio 菜单中选择“构建|部署解决方案”选项。

你也可以手工执行部署，请参考前面的章节“创建 CLR 数据库对象”。要做到这一点，你实际上需要把编译好的动态库复制到一个 SQL Server 可以访问到的目录或者文件共享位置。然后运行 CREATE ASSEMBLY 语句注册该动态库，并把它复制到数据库中。

```
create assembly usp_ImportFile
from 'C:tempusp_ImportFile.dll'
WITH PERMISSION_SET = EXTERNAL
```

CREATE ASSEMBLY 语句把 C 盘临时目录下的“usp_ImportFile.dll”文件的内容复制到了 SQL Server 数据库中。WITH PERMISSION SET 从句用来指定这个程序集可以访问 SQL Server 数据库外部的资源。这里需要这么做，因为存储过程要读取外部文件。

```
CREATE PROCEDURE usp_ImportFile
```

```
    @filename nvarchar(1024),
    @columnname nvarchar(1024) OUT
    AS
EXTERNAL NAME usp_ImportFile.[usp_ImportFile.StoredProcedures]
    .usp_ImportFile
```

CREATE PROCEDURE 语句用来使用 CLR 程序集创建一个新的 SQL Server 存储过程。这个 CLR 存储过程使用两个参数。第一个是输入参数，第二个是输出参数。EXTERNAL NAME 从句使用一个三部分组成的名称来指定动态库中的目标方法。该名称的第一部分代表了程序集名称。第二部分表示类名称。如果类被包含在命名空间下(本例中就是这样)，命名空间前缀也必须作为类名的一部分，都要放在方括号内。最后名称的第三部分指定了即将执行的方法名。

调用存储过程

在 CLR 存储过程创建完成之后，它可以像所有 T-SQL 存储过程那样被调用，请看下面的示例代码：

```
DECLARE @myColumn ntext
EXEC usp_ImportFile 'c:temptestfile.txt' @myColumn
```

(作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国)

CLR 触发器（上）

除了存储过程和用户定义函数，SQL Server 2005 中.NET 新整合的能力还提供了对创建触发器的支持。要使用 Visual Studio 2005 创建触发器，你要从前面的例子中了解到怎样创建项目。要使用 Visual Studio 2005 创建触发器，选择“新建|项目”选项，指定项目名称，点击“确定”创建项目。

在这个项目中，我使用“ti_ShowInserted”作为我的触发器名称。这个触发器实现的功能是查询插入到表中的一行，并把数据显示出来。给项目指定名称之后点击“确定”，会出现“新建数据库引用”对话框，我使用与前面的例子中相同的设置。接下来，我使用“项目|添加触发器”菜单项为 CLR 触发器创建启动项目。如图 3-10 所示：

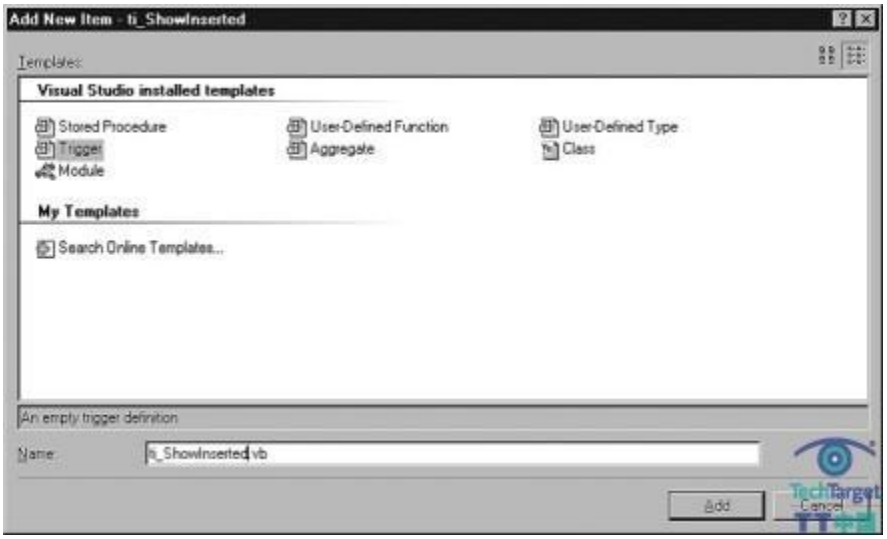

图 3-10：添加 CLR 触发器

正如你在前面的 CLR 数据库对象例子中看到的那样，你可以从模板列表中选择“触发器”选项，然后在名称文本框中提供触发器的名称。Visual Studio 2005 会生成启动项目文件，你可以给其中添加自己的代码。启动项目包括需要的导入指令，并且会生成一个类，本例中类名为“Triggers”，名为“ti_ShowInserted”的方法前面有需要的属性设置。下面的代码列表展示了名为“ti_ShowInserted”的 CLR 触发器的完整代码：

```
Imports System
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server
Imports System.Data.SqlClient

Partial Public Class Triggers
    ' Enter existing table or view for the target and uncomment
      the attribute line
    <Microsoft.SqlServer.Server.SqlTrigger(Name:="ti_ShowInserted", _
        Target:="Person.ContactType", Event:="FOR INSERT")> _
    Public Shared Sub ti_ShowInserted()
        Dim oTriggerContext As SqlTriggerContext = _
          SqlContext.TriggerContext
        Dim sPipe As SqlPipe = SqlContext.Pipe
        If oTriggerContext.TriggerAction = TriggerAction.Insert Then
            Dim oConn As New SqlConnection("context connection=true")
            oConn.Open()
            Dim oCmd As New SqlCommand("Select * from inserted", oConn)
            sPipe.ExecuteAndSend(oCmd)
        End If
    End Sub
End Class
```

本例中 CLR 触发器显示在数据库“Adventureworks”中表“Person.ContactTypes”上执行的一步插入操作插入的数据内容。代码中要注意的第一点是“ti_ShowInserted”方法的“属性”(在尖括号<>中括起来的代码)。该属性用来命名触发器，并标识触发器将应用到的表和引起触发器触发的事件。

使用 Visual Studio 2005 触发器模板初始化生成这一属性时，它前面会自动生成注释符号(就是使该行成为注释)。这是因为触发器模板不知道你想如何使用和在哪里使用触发器。为了使 Visual Studio 2005 能部署触发器，你需要把属性行前面的注释去掉，然后写上合适的属性。下表列出了 Visual Studio 2005 触发器模板可以使用的属性：

Property Name	Description
Name	The name the trigger will use on the target SQL Server system.
Target	The name of the table that the trigger will be applied to.
Event	The action that will fire the trigger. The following trigger events are supported: FOR INSERT, FOR UPDATE, FOR DELETE, AFTER INSERT, AFTER UPDATE, AFTER DELETE, INSTEAD OF INSERT, INSTEAD OF UPDATE, INSTEAD OF DELETE

在本例中，生成的触发器会命名为“ti_ShowInserted”。该触发器会应用到数据库“AdventureWorks”中表“Person.ContactType”上，触发器只有在发生插入操作时才会被触发。

触发器的主要代码都在“ti_ShowInserted”方法中。该代码示例利用了 ADO.NET 的另一个新对象“SqlTriggerContext”。SqlTriggerContext 对象提供关于触发触发器动作的信息，和触发器影响的列。“SqlTriggerContext”对象总是由“SqlContext”对象实例化的。通常，“SqlContext”对象提供关于调用者的上下文信息。在本例中尤其如此，“SqlContext”对象使代码可以访问执行触发器过程中创建的虚拟表。该虚拟表存储了能引起触发器执行的数据。

接下来，创建了“SqlPipe”对象。“SqlPipe”对象使触发器可以与外部调用者交互，在本例中它的作用是传递插入的数据值给调用者。“SqlContext”对象的“TriggerAction”属性用来判断是否触发器动作是一个插入操作。使用该属性的作用十分明确，它支持以下的值：

TriggerAction Value	Description
TriggerAction.Insert	An insert operation was performed.
TriggerAction.Update	An update action was performed.
TriggerAction.Delete	A delete action was performed.

如果“TriggerAction”属性等于“TriggerAction.Insert”，那么说明执行了插入操作，虚拟触发器表的内容就会被查出来并通过“SqlPipe”对象的“Execute”方法送到调用者那里。为了查询虚拟表的内容，需要一个“SqlConnection”对象和一个“SqlCommand”对象。这些对象都在“System.Data.SqlClient”表空间中。你应该注意，在采用 Server 端编程时，“SqlConnection”对象使用的连接串必须设置“context Connection=true”的值。然后名叫“oCmd”的“SqlCommand”对象被实例化，使用语句“Select * from inserted”从虚拟表中查询所有行和列，其中包括刚插入的值。最后，“ExecuteAndSend”方法和“SqlPipe”对象用来执行命令，并把结果送回调用者。

（作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国）

CLR 触发器（下）

部署触发器

代码创建好了以后，你既可以通过 Visual Studio 2005 的“构建|部署解决方案”选项把它部署到数据库，也可以像本章(第三章)前面用户定义函数的例子那样，先手工删除程序集,再重新创建程序集以及其他所有依赖对象。

要手工部署代码，你需要复制“ti_ShowInserted.dll”文件到 SQL Server 数据库系统或者复制到一个 SQL Server 数据库系统可以访问的共享位置，然后执行下面的 T-SQL 服务器命令：

```
Use AdventureWorks

create assembly ti_showinserted
from 'C:\temp\ti_ShowInserted.dll'
go

CREATE TRIGGER ti_ShowInserted
ON Person.ContactType
FOR INSERT
AS EXTERNAL NAME ti_ShowInserted.[ti_ShowInserted.Triggers].ti_ShowInserted
go
```

本例中假定“ti_ShowInserted.dll”文件被复制到了 SQL Server 服务器的“c:temp”目录。首先，使用“Create Assembly”语句把动态库复制到 SQL Server 数据库，然后，使用“Create Trigger”语句以及“External Name”从句创建名为“ti_ShowInserted”的触发器，并把它附加到表“Person.ContactTypes”。就像前面的例子一样，“As External Name”从句用三部分内容指定了程序集：程序集，类，方法。特别要注意类名那部分的参数。你必须把类名包括类名前面的命名空间名称都放到括号内。在本例中，程序集名是“ti_ShowInserted”。命名空间是“ti_ShowInserted”。类名是“Triggers”，方法名是“ti_ShowInserted”。

使用该触发器

CLR 触发器部署完成以后，在该基表上执行的每次插入操作都会触发该触发器。例如，下面的插入语句将给表“Person.ContactType”表添加一行数据，它会激活触发器：

```
INSERT INTO Person.ContactType VALUES(102, 'The Big Boss',
  '2005-05-17 00:00:00.000')
```

示例触发器“ti_ShowInserted”对刚插入的行执行一个查询。然后它用“SqlPipe”对象把查询结果返回给调用者。在本例中，触发器会把插入行的内容送回给调用者：

```
ContactTypeID Name                                               ModifiedDate
------------- -------------------------------------------------- --------------
21            The Big Boss                                       2005-05-17
00:00:00.000
(1 row(s) affected)
(1 row(s) affected)
```

(作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国)

用户定义类型（上）

与.NET CLR 整合给 SQL Server 2005 带来的另一个新特性是提供了用户定义类型(UDT)的功能。使用用户定义类型 UDT，你可以扩展 SQL Server 提供的原始类型，增加你的应用或者环境特有的数据类型。

在下面的例子中，你将看到怎样创建一个代表了性别代码的用户定义类型：“M”代表男性，“F”代表女性。因为你可以把该数据存储在一个标准的一个字节的字符字段里，使用用户定义类型可以确保该字段只能接受这两种值，不需要额外的触发器，限制条件或者其他数据校验技术。

使用 Visual Studio 2005 创建用户定义类型，需要先选择“新建|项目”选项，指定你的项目名称，然后点击确定创建项目。在本项目中，我给新建的用户定义类型起名为“Gender”。在给项目指定名称，并点击“确定”以后，为了部署项目到指定的 SQL Server 数据库服务器，我用需要的连接值在“新建数据库引用”对话框填好。接下来，我点击“项目|添加用户定义类型”选项，显示“添加新项目”对话框。如图 3-11 所示：

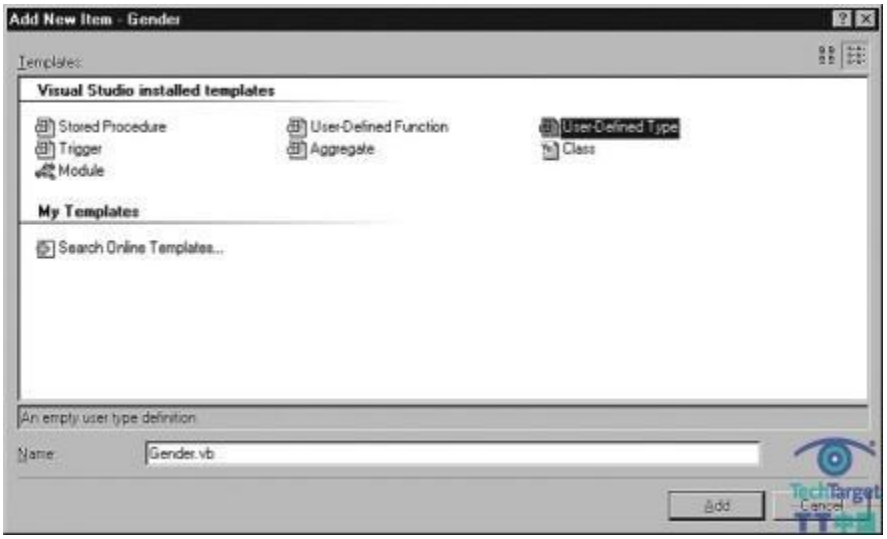

图 3-11：创建.NET SQL Server 用户定义类型

Method	Description
IsNull	This required method is used to indicate if the object is nullable. SQL Server 2005 requires all UDTs to implement nullability, so this method must always return true.
Parse	This required method accepts a string parameter and stores it as a UDT.
ToString	This required method converts the contents of the UDT to a string.
Default constructor	This required method creates a new instance of the UDT.

表 3-1：需要的用户定义类型方法

从 SQL Server 模板列表中选择“用户定义类型”。输入你想定义的类名称，点击打开，让 Visual Studio 给用户定义类型生成启动项目文件。启动项目文件实现了所有用户定义类型 SQL Server 2005 需要的四个方法。这些方法需要实现 SQL Server 用户定义类型的接口契约需求(你可以添加代码，让用户定义类型执行有意义的操作)。需要的这四个用户定义类型方法显示在了表 3-1 中。你可以看到为“M(男性)”和“F(女性)”实现的用户定义类型的完整“Gender”类。代码如下：

```
Imports System
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server
Imports System.IO

_
_
Public Structure Gender
Implements INullable, IBinarySerialize
Public Sub Read(ByVal r As BinaryReader) _
Implements IBinarySerialize.Read
m_value = r.ReadString.ToString()
End Sub
Public Overrides Function ToString() As String
If m_value.IsNull = False Then
Return m_value.Value
Else
Return Nothing
End If
End Function
Public ReadOnly Property IsNull() As Boolean _
Implements INullable.IsNull
```

```
        Get
    If m_value.IsNull = True Then
        Return True
        Else
        Return False
        End If
        End Get
    End Property
Public Shared ReadOnly Property Null() As Gender
        Get
    Dim h As Gender = New Gender
        h.m_Null = True
        Return h
        End Get
    End Property
Public Shared Function Parse(ByVal s As SqlString) As Gender
    If s.IsNull Then
        Return Null
        End If
    Dim u As Gender = New Gender
        u.Value = s
        Return u
    End Function
    ' Create a Value Property
Public Property Value() As SqlString
        Get
    Return m_value
        End Get
    Set(ByVal value As SqlString)
If (value = "M" Or value = "F") Then
        m_value = value
        Else
    Throw New ArgumentException _
("Gender data type must be M or F")
        End If
        End Set
    End Property
    ' Private members
  Private m_Null As Boolean
  Private m_value As SqlString
    End Structure
```

要创建用户定义类型，代码必须遵从一定的规则。类的属性必须是“serializable”的。类必须实现“INullable”接口，类名必须设置为用户定义类型的名称。你可以根据需要增加实现“IComparable”接口。在本例中，“Gender”是类名。在代码中靠近结尾的部分，你可以看到一个私有变量名为“m_value”，它是为保存数据类型的值而声明的。

像其他 CLR 数据库对象一样，属性在 CLR 用户定义类型的构建过程中扮演者着重要角色。SQL Server 用户定义类型属性可接受的属性值已列举在表 3-2 中。

代码中要注意的第一件事就是这里实现了“INullable”和“IBinarySerialize”接口。“INullable”接口是用户定义类型必须实现的。“IBinarySerialize”接口是用户定义类型使用“Format.UserDefined”属性所必须的。因为本例中使用了字符串数据类型，所以需要使用“Format.UserDefined”属性，这意味着这个用户定义类型还需要编写代码处理用户定义类型的序列化。在实现上来说，这意味着该类必须实现“IBinarySerialize”的“Read”方法和“Write”方法，你可以在它随后的代码段中看到这一点。

一开始，你对于使用“IBinarySerialize”接口可能似乎有一些恐惧，但是，一旦你看了“Read”和“Write”方法，它实际上就很简单了。“Read”方法简单地使用了“ReadString”方法把值赋给用户定义类型的“m_value”变量(包含了用户定义类型的值)。同样，“Write”方法使用“Write”方法序列化“m_value”变量的内容。

Property	Description
Format.Native	SQL Server automatically handles the serialization of the UDT. The Format.Native value can only be used for UDTs that contain fixed-sized data types. The following data types are supported: bool, byte, sbyte, short, ushort, int, uint, long, ulong, float, double, SqlByte, SqlInt16, SqlInt32, SqlInt64, SqlDateTime, SqlSingle, SqlDouble, SqlMoney. If this property is used, the MaxByteSize property cannot be used.
Format.UserDefined	The UDT class is responsible for serializing the UDT. The format. UserDefined value must be used for variable-length data types like String and SQLString. If this value is used, the UDT must implement the IBinarySerialize interface and the Read and Write routines. If this property is used, the MaxByteSize property must also be specified.
MaxByteSize	Specifies the maximum size of the UDT in bytes.
IsFixedLength	A Boolean value that determines if all instances of this type are the same length.
IsByteOrdered	A Boolean value that determines how SQL Server performs binary comparisons on the UDT.
ValidationMethodName	The name of the method used to validate instances of this type.
Name	The name of the UDT.

表 3-2：用户定义类型 Attribute 属性

“ToString”方法检查看“m_value”变量的内容是否为 null。如果是 null 的话，那么就会返回字符串“null”。否则，“m_value”的“ToString”方法返回内容的字符串值。

接下来的代码段定义了“IsNull”属性。该属性的 get 方法检查“m_value”变量的内容，如果“m_value”为“null”，则返回值 true。否则，get 方法返回 false。接下来，你可以看到 Null 方法，它是模板为满足用户定义类型对 null 的需求而生成的。

“Parse”方法接受字符串参数，它存储在对象的 Value 属性中。你可以看到在下面一点的代码中看到 Value 属性的定义。“Parse”方法必须声明为静态 static 的，或者如果你用的是 VB.NET，它必须声明为共享 Shared 属性。

“Value”属性是本示例实现特有的。在本例中，“Value”属性用来存储和查询用户定义类型的值。它还负责编辑容许值。在 set 方法中，你可以看到只有值“M”和“F”是允许的。如果使用了其他值，会引发并抛出异常通知调用者“Gender 数据类型的值只能是‘M’或者‘F’”。

(作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国)

用户定义类型（下）

部署用户定义类型

与 CLR 存储过程或者函数非常相似，代码编写完成之后，用户定义类型被编译为动态库。该动态库可以用“CREATE ASSEMBLY”和“CREATE TYPE”语句导入到 SQL Server 中，或者也可以简单地使用 Visual Studio 2005 的部署选项。你可以看到手工创建 CLR 用户定义类型的 T-SQL 代码。如下所示：

```
create assembly Gender
from 'C:tempGender.dll'
go
CREATE TYPE Gender
EXTERNAL NAME Gender.[Gender.Gender]
go
```

这段代码假定“gender.dll”已经复制到 SQL Server 服务器的“c：temp”目录下。在“CREATE TYPE”语句中要注意的一点是类参数。在前面的 CLR 示例中，“External Name”从句的第一部分指定了要使用的程序集。在用户定义类型里，名称的第二部分指定命名空间和类。在这个 Gender 示例中，命名空间是“Gender”，用户定义类型的类名也是“Gender”。

使用用户定义类型

一旦用户定义类型创建好了，你就可以用 T-SQL 使用它，用法与 SQL Server 的原生数据类型很像。然而，因为用户定义类型包含方法和属性，所以用起来还有一些差异。下面的例子展示了 Gender 用户定义类型被用作变量的用法和访问 Value 属性的用法。

```
DECLARE @mf Gender

SET @mf='N'

PRINT @mf.Value
```

在代码中用户定义类型的变量是使用标准 T-SQL 的“DECLARE”语句，“SET”语句用来给用户定义类型的“Value”属性赋值“N”。因为“N”不是一个有效值，所以会出现下面的错误：

```
.Net SqlClient Data Provider: Msg 6522, Level 16, State 1, Line 2

A CLR error occurred during execution of 'Gender':

System.ArgumentException: Gender 数据类型的值只能是 'M' 或者 'F'

at Gender.set_Value(SqlString value)
```

就跟用户定义类型可以用做变量一样，用户定义类型还可以被用来创建列。下面的代码展示了使用“Gender”用户定义类型创建表的方法：

```
CREATE TABLE MyContacts
(ContactID int,
FirstName varchar(25),
LastName varchar(25),
MaleFemale Gender)
```

虽然用用户定义类型创建列与使用原生数据类型用法相同，但是给用户定义类型赋值时与标准列赋值有一点不同。复杂的用户定义类型可以包含多个值。如果是那样的话，你需要赋值给用户定义类型的成员。你可以通过给成员前面加前缀符号(.)来访问用户定义类型的成员。在本例中，因为用户定义类型只是一个简单的值，你可以像给内建的数据类型赋值那样给它赋值。下面的语句展示了怎样给包含“Gender”用户定义类型的示例表“MyContacts”插入一行。

```
INSERT INTO MyContacts VALUES(1, 'Michael', 'Otey', 'M')
```

要使用“SELECT”语句查询用户定义类型的内容，你需要使用“用户定义类型.成员”引用用户定义类型的列来查询，如下面的语句所示：

```
SELECT ContactID, LastName, MaleFemale.Value FROM MyContacts
```

如果你想查看数据库中已经创建了那些用户定义类型，你可以查询“sys.Types”视图。查询语句如下所示：

```
SELECT * FROM sys.Types
```

(作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国)

用户定义函数

.NET CLR 集成还有另一个新特性：那就是可以创建基于.NET 的用户定义函数(UDF，user-defined function)。返回标量类型的用户定义函数必须返回一个.NET 数据类型，该.NET 数据类型可以隐式地转换为 SQL Server 数据类型。用.NET Framework 编写的纯量函数在某些情况下使用起来比 T-SQL 更方便，因为它跟 T-SQL 函数不一样，.NET 函数是用编译的代码创建的。用户定义函数还可以返回 table 类型，在这种情况下，函数必须返回一个结果集。

要使用 Visual Studio 2005 创建用户定义函数(UDF)，需要先选择“新建|项目”选项，然后选择 SQL Server 项目模板。如下图 3-7 所示：

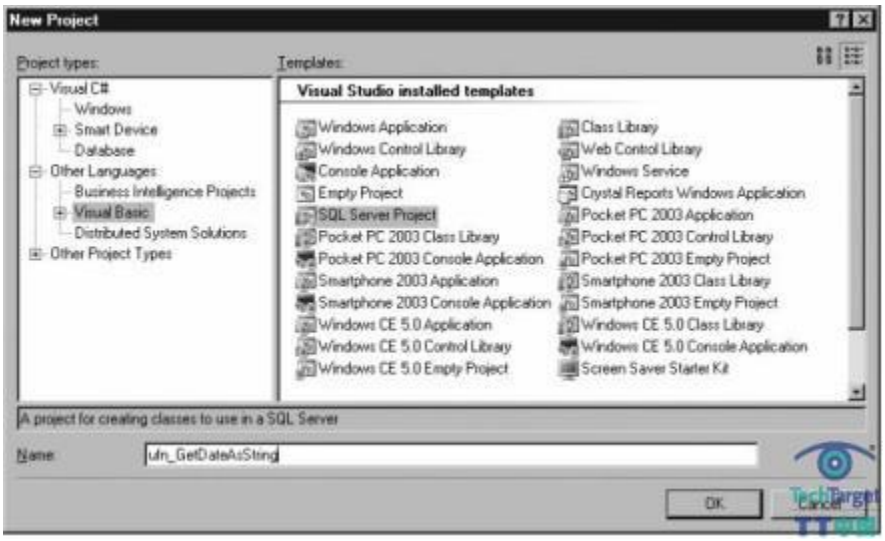

图 3-7：创建一个新的 SQL Server 用户定义函数项目

就像前面展示的存储过程的例子一样，首先指定项目名称，然后点击确定创建项目。在本例中，如下图 3-7 所示，你可以看到我用“ufn_GetDateAsString”作为我的用户定义函数名称。该函数返回一个包含系统日期和时间的字符串值。在指定了项目名称之后，点击“确定”就会显示 CLR 函数项目的“新建数据库引用”对话框，如图 3-8 所示：

注意：当已经存在数据库引用时，显示的就是“添加数据库引用”对话框，而不是“新建数据库引用”对话框。如果你在创建了“usp_ImportFile ”项目之后立即就创建“ufn_GetDateAsString”函数，就会出现这种情况。

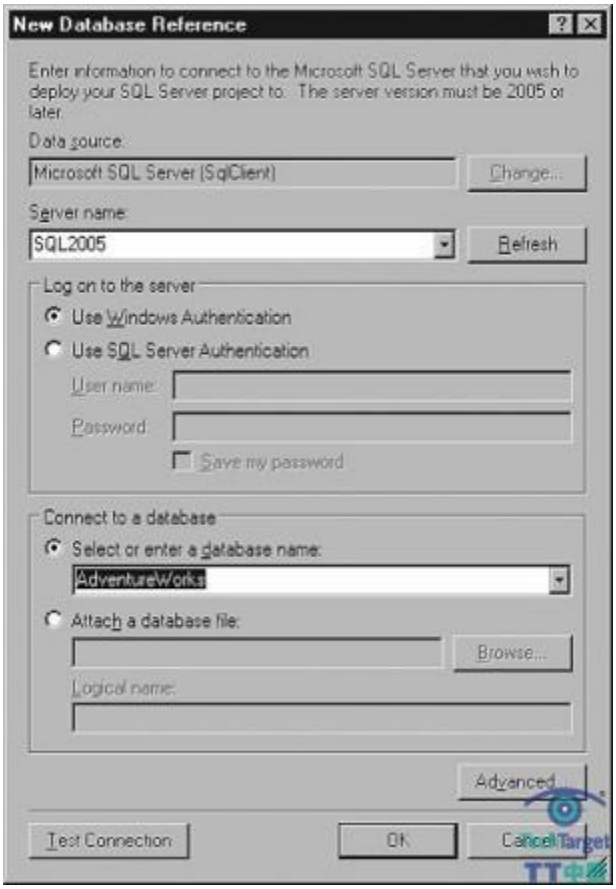

图 3-8：新建数据库引用对话框

“新建数据库引用”对话框定义了 Visual Studio 项目和 SQL Server 之间的连接。该项目会连接到名为“sql2005”的 SQL Server 系统，该函数将被部署到 AdventureWorks 数据库。

Visual Studio 项目创建了，连接也定义了，接下来你就可以点击“项目|添加函数”菜单选项，显示“添加新项目”对话框。如图 3-9 所示：

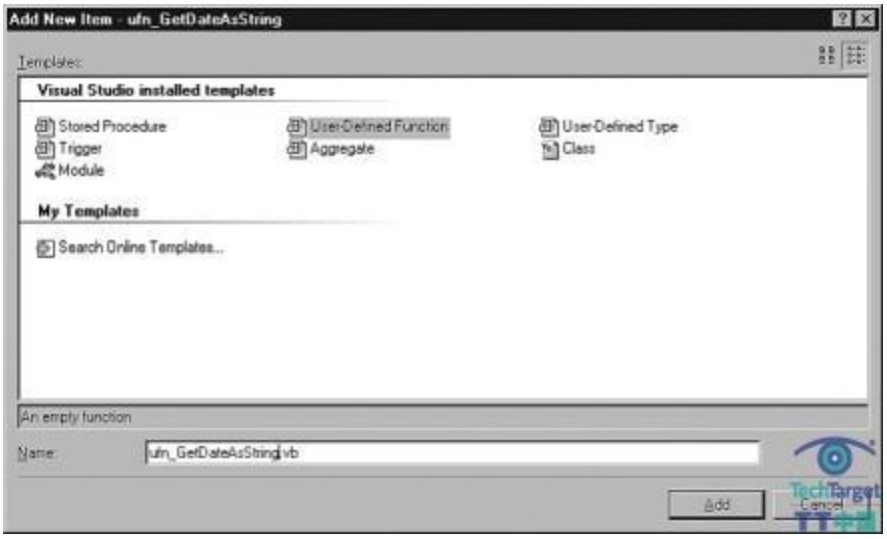

图 3-9：添加 CLR 用户定义函数

Visual Studio 使用“SQL Server 函数”项目模板创建了一个启动项目，其中包含对 SQL Server .NET 数据提供器的引用和你源代码的基本函数包装。你可以根据需要添加剩下的代码。下面的代码就是完整的 CLR 函数“ufn_GetDateAsString”，该函数执行一个日期转换为字符串的基本功能：

```
Imports System
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server
Partial Public Class UserDefinedFunctions
    _
Public Shared Function ufn_GetDateAsString() As SqlString
Dim dtDataTime As New DateTime
Return dtDataTime.ToString()
End Function
```

这里不需要 Microsoft.SqlServer.Server 命名空间，因为这个函数不执行任何数据访问的功能。接下来，Visual Studio 2005 会生成“UserDefinedFunctions”类，其中包含了本程序集将作为用户定义函数展示的所有方法。你还可以看到用来把“GetDateAsString”方法标记为用户定义函数的属性。本段代码只是一个简单的示例，只是把系统时间转换为字符串数据类型，然后返回给调用者。

部署该函数

要在 SQL Server 数据库中创建函数，首先必须像你在前面的存储过程的例子中看到的那样创建程序集。如果你用的是 Visual Studio 2005，那么你可以简单地选择“构建|部署解决方案”选项，一切就完成了。

如果你是手工来做，你需要把“ufn_GetDataAsString.dll”文件复制到某个位置(SQL Server 系统要可以访问到该位置)，然后创建关于该函数的程序集。下面的 CREATE ASSEMBLY 语句可以用来把“ufn_GetDateAsString.dll”的内容复制到 SQL Server 数据库：

```
CREATE ASSEMBLY ufn_GetDataAsString

FROM 'MyFileShareCode Libraryufn_GetDataAsString.dll'
```

然后使用“CREATE FUNCTION”语句创建新的在程序集中执行相应方法的 SQL Server 函数。下面的代码段展示了使用“CREATE FUNCTION”语句创建.CLR 用户定义函数的方法：

```
CREATE FUNCTION ufn_GetDateAsString()

RETURNS nvarchar(256)

EXTERNAL NAME

ufn_GetDateAsString.UserDefinedFunctions.ufn_GetDateAsString
```

对于用户定义函数，“CREATE FUNCTION”语句扩展使用了“EXTERNAL NAME”从句，它本质上就是连接了用户定义函数名与.NET 程序集中相应的方法。在本例中，“fn_GetDateAsString”函数使用名为“fn_GetDateAsString”的程序集。在程序集内部，它使用的是“UserDefinedFunctions”类和类里的“ufn_GetDateAsString”方法。

使用该函数

函数创建以后，它可以被常规 SQL Server 函数调用。在下面的例子中你可以看到怎样执行 GetDateAsString 函数。

```
SELECT dbo.GetDateAsString()
```

```
End Class
```

(作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国)

调试 CLR 数据库对象

在.NET Framework，Visual Studio 2005 和 SQL Server 2005 的集成功能中，最酷的特性之一就是支持调试你创建的 CLR 数据库对象的能力。如此紧密的整合，使得 SQL Server 在与像 Oracle 和 DB2 这类也支持用.NET 代码创建存储过程和函数的数据库的竞争中遥遥领先。虽然其他数据库产品也提供创建这些对象的功能，但是它们不支持集成调试的能力。Visual Studio 2005 支持你在 CLR 数据库对象中设置断点，然后无缝地单步执行你的代码，执行所有你能在标准 Windows 或者 Web 应用程序中执行的调试任务，包括设置断点，单步执行代码，监视和改变变量的值，创建监视变量(甚至在 T-SQL 与 CLR 代码之间进行调试)。Visual Studio 2005 会自动生成测试脚本，并添加到你的项目中。你可以定制并使用这些测试脚本来执行你创建的 CLR 数据库对象。

要使用 Visual Studio 2005 调试 SQL Server 项目，首先打开你想要调试的项目，然后转到“服务器”窗口，在其中右击数据库连接。在弹出菜单中选择“允许 SQL/CLR 调试”选项。如下图 3-13 所示：

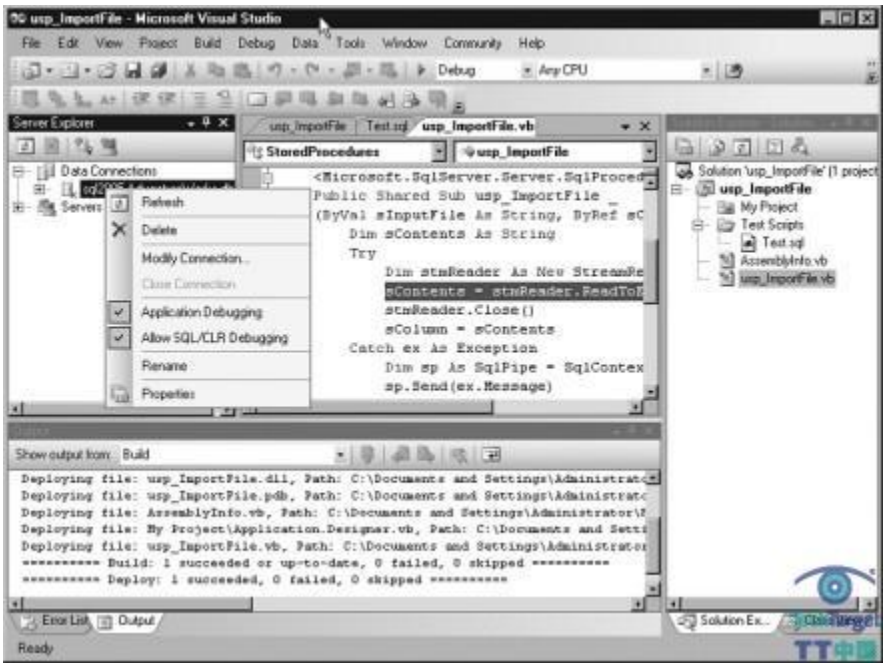

图 3-13：设置“允许 SQL/CLR 调试”选项

接下来，设置你想用来运行数据库对象的脚本。使用“解决方案”窗口，打开“测试脚本”文件夹，然后打开“Test.sql”文件。你可以准备多个测试脚本，但是“Test.sql”脚本是默认提供的。如果你想修改该 Visual Studio 2005 用来运行 CLR 数据库对象的脚本，你只需要简单地右击“测试脚本”文件夹下列出的目标脚本，然后选择“设为默认调试脚本”选项就可以了。如下图 3-14 所示：

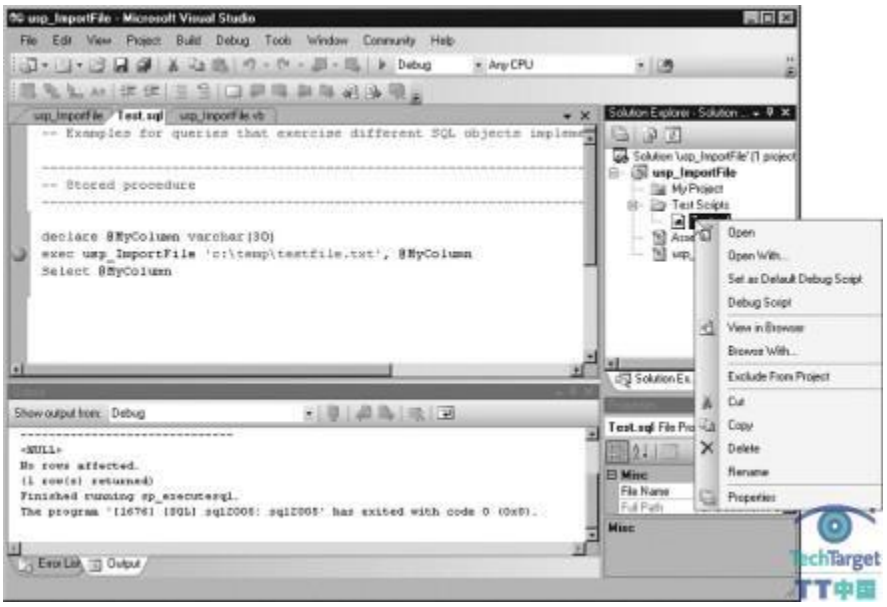

图 3-14：设置默认调试脚本

要使用默认的“Test.sql”脚本，请使用 Visual Studio 2005 编辑器打开该文件。这里你可以看到 T-SQL 测试每一种不同的 CLR 数据库对象类型的样板文件代码。跳到你需要的代码段，然后编辑代码执行数据库对象。你可以看到“usp_ImportFile”存储过程的测试代码示例如下：

```
-- Examples for queries that exercise different SQL objects
-- implemented by this assembly
------------------------------------------------------------------

-- Stored procedure
------------------------------------------------------------------

declare @MyColumn varchar(30)
```

```
exec usp_ImportFile 'c:temptestfile.txt',@MyColumn
Select @MyColumn
```

测试脚本准备好了以后，用 Visual Studio 的“调试|启动”选项，或者直接敲击“F5”就可以运行“Test.sql”，它会执行你的 CLR 数据库对象。你可以在下图 3-15 中看到使用 Visual Studio 2005 调试器单步调试 SQL Server 项目的例子。

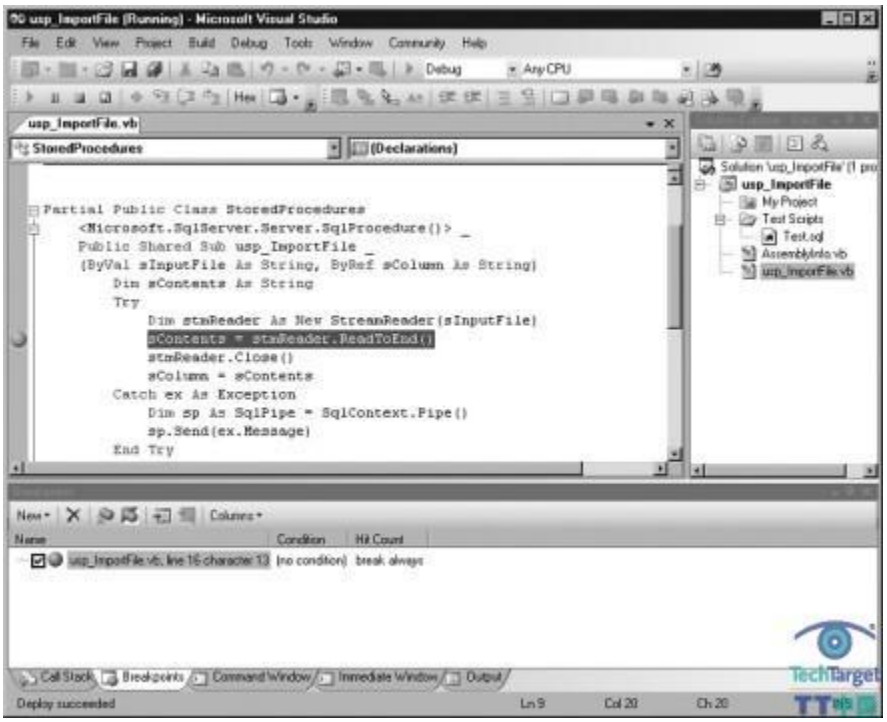

图 3-15：调试 Visual Studio 2005 SQL Server 项目

在这一步中，你可以单步调试代码，设置新断点，还可以修改和监视变量的值。

（作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国）

CLR 聚合功能（上）

CLR 聚合是 SQL Server 2005 引入的另一新的.NET 数据库对象类型。本质上，自定义聚合功能是处理查询过程中对聚合数据分组用法的一个扩展功能。SQL Server 一直提供有我们可以在查询中使用的聚合功能，例如：MIN，MAX 和 SUM。自定义聚合功能使我们可以用自定义的聚合功能扩展原来的聚合功能。CLR 聚合是一个非常好用的功能，支持创建对 CLR 自定义类型的聚合函数。与原生聚合函数一样，自定义聚合支持你对一组数据执行计算并返回单个值的功能。在创建 CLR 聚合时，你需要提供执行聚合的功能逻辑。在本节中，你将看到怎样创建一个简单的聚合功能，计算一组数字的最大差值。

要使用 Visual Studio 2005 创建聚合功能，你要点击“新建|项目”选项，指定项目名称，然后点击“确定”创建项目。本例中使用的名称为“MaxVariance”。在给项目命名，点击确定之后，使用你需要的连接配置完成“新建数据库引用”对话框，指向你的 SQL Server 服务器和数据库。接下来，为了创建聚合功能，我点击“项目|添加聚合”选项，显示“添加新项目”对话框。如图 3-12 所示：

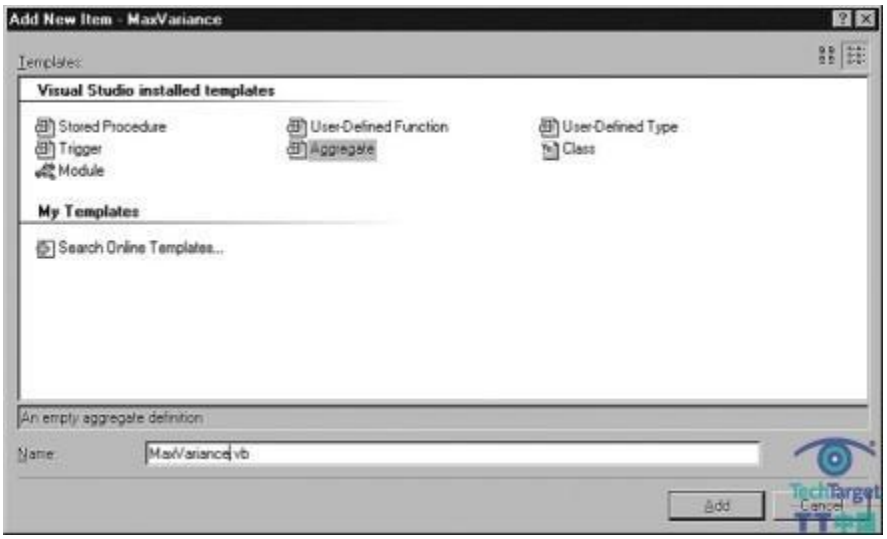

图 3-12：创建 CLR 聚合功能

在 SQL Server 模板列表中选择“聚合 Aggregate”，然后输入类名，点击“确定”。如图 3-12 所示：我使用的名称是“MaxVariance”。Visual Studio 会给聚合类生产启动项目。与自定义类型相似，SQL Server CLR 聚合功能模板实现了 SQL Server 2005 对所有 CLR 聚合功能需要的四个方法。这四个必须实现的方法都列举在了下表 3-3 中。

Method	Description
Init	This required method initializes the object. It is invoked once for each aggregation.
Accumulate	This required method is invoked once for each item in the set being aggregated.
Merge	This required method is invoked when the server executes a query using parallelism. This method is used to merge the data from the different parallel instances together.
Terminate	This required method returns the results of the aggregation. It is invoked once after all of the items have been processed.

表 3-3：必须实现的聚合功能方法

你可以看到实现“MaxVariance”聚合功能的代码。如下所示：

```
Imports System
Imports System.Data
Imports System.Data.Sql
Imports System.Data.SqlTypes
Imports Microsoft.SqlServer.Server

_
_
Public Structure MaxVariance
    Public Sub Init()
        m_LowValue = 999999999
        m_HighValue = -999999999
    End Sub
    Public Sub Accumulate(ByVal value As Integer)
        If (value > m_HighValue)
            m_HighValue = value
        End If
        If (value < m_LowValue)
            m_LowValue = value
        End If
    End Sub
    Public Sub Merge(ByVal Group as MaxVariance)
        If (Group.GetHighValue() > m_HighValue)
            m_HighValue = Group.GetHighValue()
```

```
                End If
            If (Group.GetLowValue() < m_LowValue)
                m_LowValue = Group.GetLowValue()
                End If
                End Sub
        Public Function Terminate() As Integer
            return m_HighValue - m_LowValue
                End Function
                ' Helper methods
        Private Function GetLowValue() As Integer
            return m_LowValue
                End Function
        Private Function GetHighValue() As Integer
            return m_HighValue
                End Function
            ' This is a place-holder field member
                Private m_LowValue As Integer
                Private m_HighValue As Integer
                End Structure
```

在这段代码的顶部，你可以看到标准的导入语句集，它们都是 CLR 对象需要使用的，下面是 CLR 聚合功能对象需要使用的序列化属性。在那些代码后面，在“Init”方法内有两个参数，“m_LowValue”和“m_HighValue”，分别被赋予高值和低值，确保它们从列表中得到赋值。这两个变量是在代码中靠近底部的位置声明的，它们用来保存聚合程序遇到的最小值和最大值。“Init”方法只被调用一次，就是在对象第一次初始化的时候被调用。

虽然“Init”方法只被调用一次，但“Accumulate”方法会对结果集中的每一行调用一次。在本例中，“Accumulate”方法把新值与存储在“m_HighValue”和“m_LowValue”中的值相比较。如果新值比当前较高的值更大，就把新值存储在“m_HighValue”变量中。如果该新值比“m_LowValue”的值要小，就把该值存储在“m_LowValue”变量中。

“Merge”方法用在聚合功能并行处理的情况，一般大部分查询都不是如此。如果“Merge”方法被调用，它的任务是从并行实例中导入当前聚合功能值。在本例中你可以看到，它使用两个帮助函数(本质就就是导出“m_HighValue”和“m_LowValue”变量的值)来做到这一点。

“Terminate”方法在所有结果被处理完之后调用一次。在本例中，“Terminate”方法简单地把找到的最大值减去找到的最小值，把它们的差值返回给调用者。

(作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国)

CLR 聚合功能（下）

部署聚合功能

把该类编译成动态链接库之后，你可以使用两种方法将动态链接库导入为 SQL Server 的程序集。使用 Visual Studio 2005 的部署选项或者手工使用“CREATE ASSEMBLY”语句和“CREATE AGGREGATE”语句。手工导入的语句示例如下：

```
create assembly MaxVariance
from 'C:tempMaxVariance.dll'
go
CREATE AGGREGATE MaxVariance (@maXVar int)
RETURNS Int
EXTERNAL NAME MaxVariance.[MaxVariance.MaxVariance]
go
```

像前面的例子一样，上面的脚本假定“maxvariance.dll”文件已经被复制到了 SQL Server 服务器本地的“c：temp”目录。在“CREATE AGGREGATE”语句和“EXTERNAL NAME”从句中，名称的第一部分指定了要使用的程序集名称，名称的第二部分指定了命名空间和类名。这里所有的值被命名为“MaxVariance”。

使用聚合功能

你可以使用该聚合功能，与 SQL Server 内建的聚合函数用法一样。有一点小小的差异是，自定义聚合函数需要加上 schema 前缀，这样系统才能找到它。下面的语句就用到了“MaxVariance”聚合功能：

```
SELECT dbo.MaxVariance(MinQty) FROM Sales.SpecialOffer
```

该语句的执行结果显示了“Sales.SpecialOffer”这一列中最大值和最小值之间的差值。如下所示：

```
-----------
```

```
61

(1 row(s) affected)
```

(作者：McGraw-Hill 译者：冯昀晖 来源：TT 中国)