在 Visual FoxPro 中使用 wwDotnetBridge 调用 .NET 组件

作者：Rick Strahl
http://www.west-wind.com
mailto:rstrahl@west-wind.com
翻译：xinjie 2018.05.08

目录

微软的 .NET Framework 已经存在了10年并一直在微软平台上稳步发展。.NET 现在进入了Windows 的大多数方面，并且针对桌面系统它已经是相当的标准。随着 Windows 的发展，.NET 代码也可以充分利用 Windows 的系统特性，取代之前对于 Windows API 的过度依赖。

作为一个 FoxPro 开发者，你或许不会立刻使用 .NET ，但是你会惊讶的发现在 .NET Framework Runtimes 存在很多有用的功能。它甚至不涉及作为 Windows 间接组成部分的 Microsoft System SDKs、第三方的 toolkits 或者其他开源的库。在你的 FoxPro 代码中，你可以随时应用大量的 .NET 功能!

在这里，我将简单介绍 .NET 的 COM 互操作（COM Interop）。COM 互操作是内置于 .NET 的一个特性，此特性允许通过 COM 访问外部内容并可以将其作为一个 COM 服务。通过 COM 互操作，你将拥有在 FoxPro 应用程序中访问 .NET 内容的能力。

这方面我已经写了很多的文章，如果你想更深入的了解 .NET 中的 COM 互操作，我建议你阅读下面这篇文章：

在 Visual FoxPro 中使用 .NET COM 组件

本文仅仅涵盖了传统的 COM 互操作，它非常的简单。本文真正的主角是一个库：wwDotnetBridge。它通过许多的附加功能大大的扩展了 COM 互操作。

wwDotnetBridge 是自由且开源的。你可以在你的应用程序中任意的使用它。你可以在 GitHub 或者项目主页找到最新的包含源代码的版本：

- wwDotnetBridge Home Page
- wwDotnetBridge on GitHub（英文）
- wwDotnetBridge on GitHub（中文）

让我们勇往直前！

COM 互操作快速预览

在详细了解 wwDotnetBridge 之前，我们还是先看看 .NET 的 COM 互操作基础知识，了解一下哪些是 .NET 提供的，哪些不是。

.NET 提供了 COM 的互操作性，它允许你从 .NET 中调用 COM 组件，同样也允许你通过 COM 来调用 .NET 组件。这里我仅仅讨论下后者——从 Visusl FoxPro 代码中访问 .NET 组件。

对于要向 COM 公开的 .NET，创建的 COM 组件必须注册为 .NET COM 对象。这意味着必须使用启用 COM 注册的方式编译它们，并且必须将它们标记

为对 COM 可见。

创建. NET 类时，您可以选择指定要将该类型导出到 COM。一旦组件被编译，必须通过一个名为 RegAsm 的特殊工具进行注册，以便 COM 可访问。

为 COM 编译的. NET 对象在注册表中注册时和正常的 COM 对象对象一样，但它们包括其他注册表项。

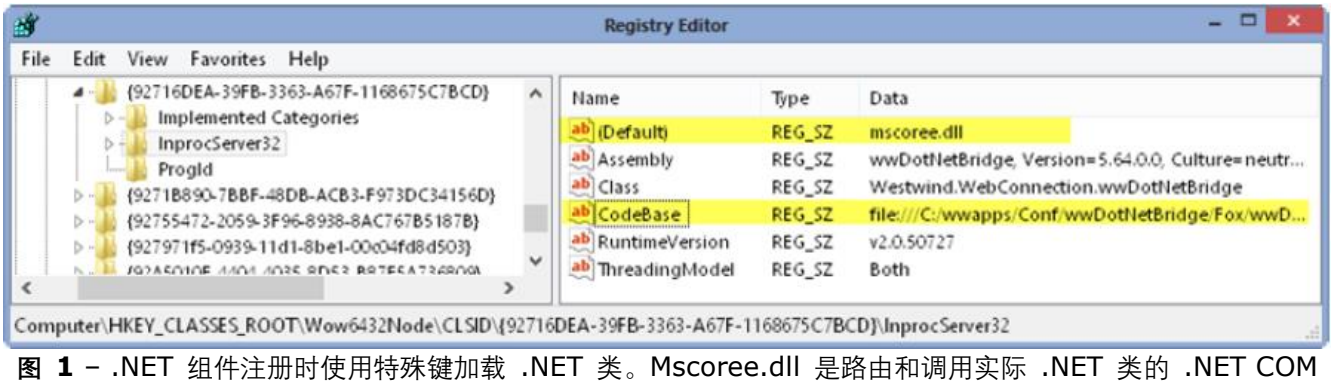


图 1 – .NET 组件注册时使用特殊键加载 .NET 类。Mscoree.dll 是路由和调用实际 .NET 类的 .NET COM 代理。

调用. NET COM 对象时，它实际上会调用类似于 COM 代理的 mscoree.dll。基于注册表项,. NET 运行时计算出要实例化的类，然后将所有 COM 调用路由到此类。

图 2 – *.NET COM 访问调用 .NET 代理，然后将 COM 调用转发到 .NET 类。*

创建 .NET COM 对象相当容易，但这种 COM 访问方案的一个主要缺点是必须将 .NET 对象显式导出到 COM 以使其可以访问 COM。 很少有本地 .NET 组件被导出到 COM。

这使 COM 互操作的这种机制主要用于编译自己的 .NET 类，并将它们编译并注册到 COM。 这不是访问组件的通用机制。

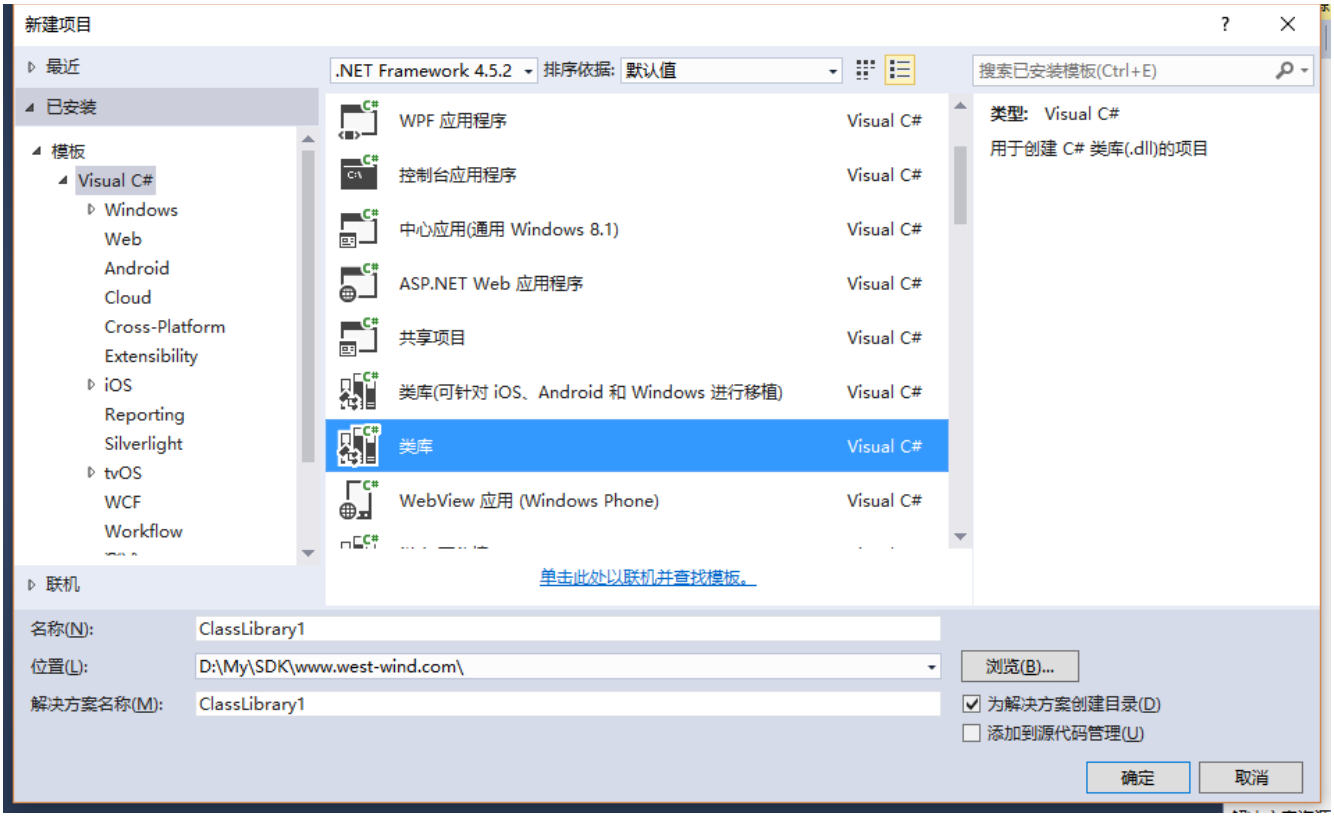
创建一个 .NET 组件并在 Visual FoxPro 中调用它

所以，我们看看如何快速的创建一个 .NET COM 组件并在 Visual FoxPro 中调用它。

本文我将使用 Visual Studio 2012（中文版使用 VS2015），但是任何版本的 VS 都可以被使用。你最好使用 .NET 4.0，但是所有的东西都可以在更新的版本中被找到。

请遵循下面的步骤：

- 打开 Visual Studio
- 创建一个新项目，并命名它为 InteropExamples
- 选择 Visual C#，并选择类库
- 命名类为 Examples.cs



现在，打开 Examples.cs 文件并增加一个简单的 .NET 类，就像这样：

C# - 我们的第一个为 COM 互操作准备的类

```
using System;
using System.Runtime.InteropServices;
namespace InteropExamples
{
    [ComVisible(true)]
```

```
[ClassInterface(ClassInterfaceType.AutoDual)]
[ProgId("InteropExamples.Examples")]
public class Examples
{
    public string HelloWorld(string name)
    {
        return "It's a helluva World, " + name;
    }
    public decimal Add(decimal number1, decimal number2)
    {
        return number1 + number2;
    }
}
}
```

接下来，通过选择解决方案资源管理器中的项目节点，确保将项目配置为自动导出标记为 COM 引用，右键单击然后选择 "属性"。转到 "生成" 选项卡并选中 "注册为 COM 互操作" 复选框。

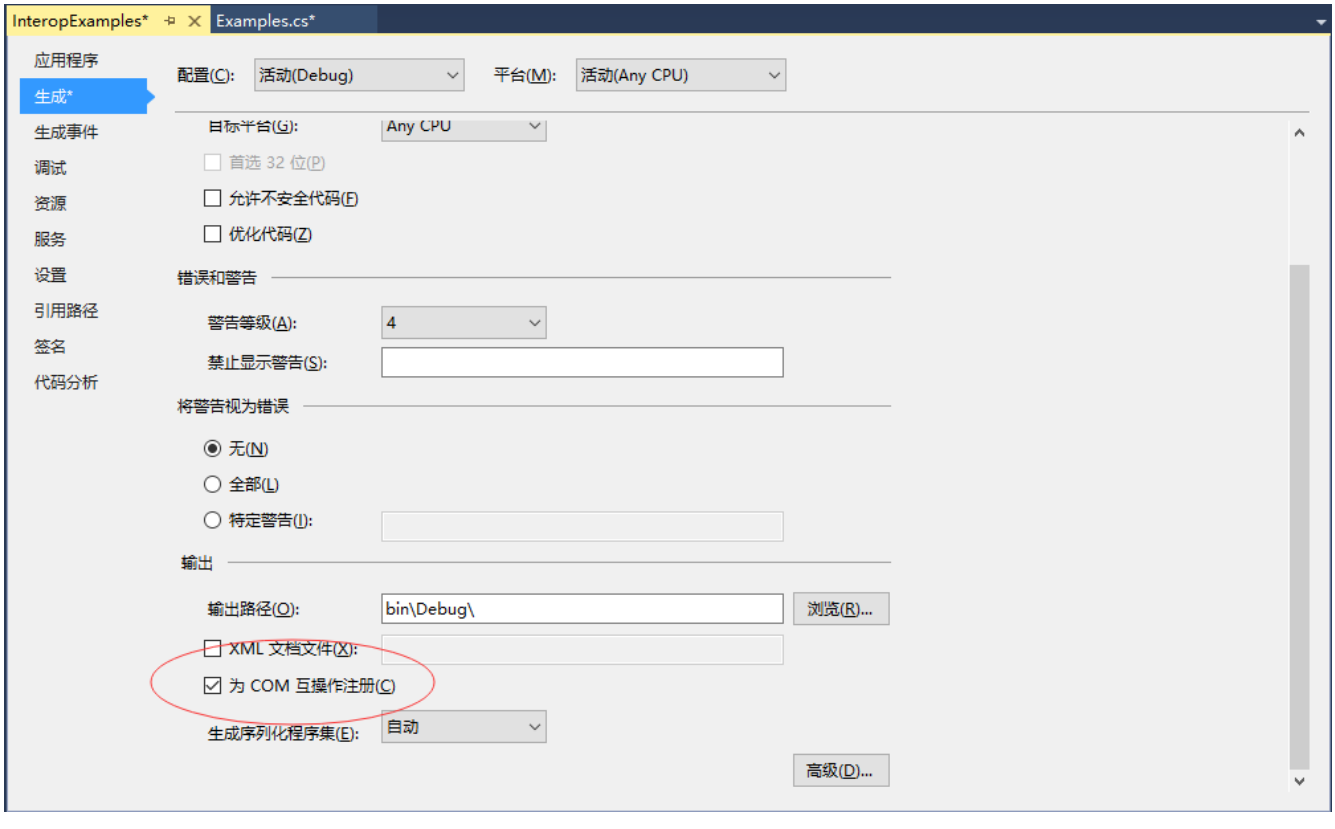


图 3 – *为了使 com 对象能够被 com 访问，它需要在您的计算机上注册。*

为了使您的组件能够通过 COM 访问，它需要在您的开发计算机上注册，并且使用图3中的复选框最容易做到这一点。但是，当您部署到客户端时，您需要使

用. NET RegAsm 实用程序注册此组件。RegAsm 生活在. NET 框架目录中，应该像这样运行：

```
RegAsm "c:\dev\ComInteropExamples.dll" /codebase
```

如果需要，这可以作为安装程序的一部分完成，某些安装程序会为您明确地执行此操作，但这很棘手，因为您必须为RegAsm找到正确的版本。

有关此过程的更多信息以及一个为您自动化执行此过程的 FoxPro 帮助可以在旧的 COM 互操作文章中找到。还有一个registercomponent.prg，演示了如何从Visual FoxPro代码执行注册。

在 FoxPro 中使用 .NET COM 组件

现在该组件已经被编译和注册，在Visual FoxPro中使用它非常简单：

```
o = CREATEOBJECT("InteropExamples.Examples")
? o.HelloWorld("Southwest Fox")
? o.Add(10,20)
```

COM 对象的名称是我在 .NET 类中指定的 [ProgId] 属性的名称。 如果我省略这个 ProgId，默认的是 .NET namespace.classname。 在这种情况下，ProgId 属性不是必需的，因为名称是相同的。 正如你所看到的，创建一个COM对象并从FoxPro使用它很容易

让我们通过返回一个对象来为类添加更多的功能。 让我们在 Person.cs 文件中创建两个.NET类：

C# - 用于示例的一组小型商业实体

```
using System;
using System.Runtime.InteropServices;

[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class Person
{
```

```
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Address Address { get; set; }
    public DateTime Entered { get; set; }

    public Person()
    {
        Address = new Address();
    }
}

[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class Address
{
    public string Street { get; set; }
    public string City { get; set; }
    public string Zipcode { get; set; }
    public string CountryCode { get; set; }
    public string Country { get; set; }
}
```

这些类是嵌套的，address 是 Person 类中的一个属性。接下来，让我们（在 Examples.cs 中）创建一些可以创建、更新和读取.NET 组件中的人员的方法：

C# - 针对 Person 类的简单的对象访问方法

```
        public List<Person> Persons { get; set; }
        public Person GetNewPerson()
        {
            return new Person();
        }
        public bool SavePerson(Person person)
        {
            if (person == null)
                return false;
            if (Persons == null)
                Persons = new List<Person>();
            if (person.Id != 0)
            {
                var matched = Persons.Where(p => p.Id == person.Id)
                .FirstOrDefault();
                if (matched != null)
                {
                    matched = person;
                    return true;
                }
            }
            Persons.Add(person);
```

```
        return true;
    }
    public Person GetPerson(int id)
    {
        Person person = Persons.Where(p => p.Id == id)
        .FirstOrDefault();
        return person;
    }
```

该代码设置了一个人员列表（使用泛型的List <Person>），其中包含我添加的任何人员。 这将是我们的'数据存储'，它在对象处于范围内时暂时保存 Person 对象。

GetNewPerson() 给我一个空的人员记录，我可以在 FoxPro 中完成填写；一个SavePerson() 方法来保存数据；GetPerson() 让我检索一个先前保存的人员记录。 它就像一个迷你数据存储库。

现在编译代码。 如果你仍然打开着 FoxPro，你可能会发现无法编译 .NET DLL，因为它是锁定的—FoxPro 将 DLL 加载到内存中，并且在加载时编译器无法更新磁盘上的物理文件。 为了让编译工作，你必须关闭 Visual FoxPro。

编译完成后，我们编写一些代码以使用FoxPro中的示例代码：

FoxPro – 访问Person Repository功能非常简单

```
LOCAL loFox as InteropExamples.Examples
loFox = CREATEOBJECT("InteropExamples.Examples")
loPerson = loFox.GetNewPerson()
loPerson.Id = 1
loPerson.FirstName = "Rick"
loPerson.LastName = "Strahl"
loPerson.Address.Street = "32 Kaiea Place"
loPerson.Entered = DATETIME()
loFox.SavePerson(loPerson)
loPerson = loFox.GetNewPerson()
loPerson.Id = 2
loPerson.FirstName = "Markus"
loPerson.LastName = "Egger"
loPerson.Address.Street = "213 Mud Lane"
loPerson.Entered = DATETIME()
loFox.SavePerson(loPerson)
loPerson = null
loPerson = loFox.GetPerson(1)
? loPerson.FirstName + " " + loPerson.Address.Street
```

代码非常简单，正如您所期望的那样。 我创建了一个将 .NET 对象传递给 FoxPro 的新 Person 记录。 我在 FoxPro 中使用数据填充 .NET 对象，然后使用原始实例调用 SavePerson() 以更新 .NET 中的数据。 第二条记录我重复了这个步骤。

最后两行通过在 Persons 集合中搜索 Person 对象列表并通过 ID 返回一个匹配或null来检索一个客户。 这一切都按预期工作。 正如你所看到的，它非常直接地在 FoxPro 中访问了 .NET 对象。

运行过程中的问题

到目前为止还不错，但是现在让我们来看看那些不适用于 COM 互操作的东西：注意，loFox 对象具有一个 Persons 属性，它的类型恰好是 List<Person>。这是一个 .NET 泛型类型，这意味着编译器修复了在编译时创建的List对象的类型。泛型是一个编译器模板工具，它使用泛型参数（本例中为 Person）作为模板创建定制类 - 本质上，编译器在此创建一个带有 Person 元素的自定义列表类。这是一个非常强大和流行的功能，用于创建强类型的泛型类型，如列表，字典甚至诸如业务对象或与其他子类型一起工作的其他对象。

问题是当使用普通的 COM 互操作时，COM 不会封送泛型类型。问题是这些类型是半动态的，并且不会导出到 COM。如果你使用命令行来访问该对象并输入loFox。您会看到"人员"列表未显示：

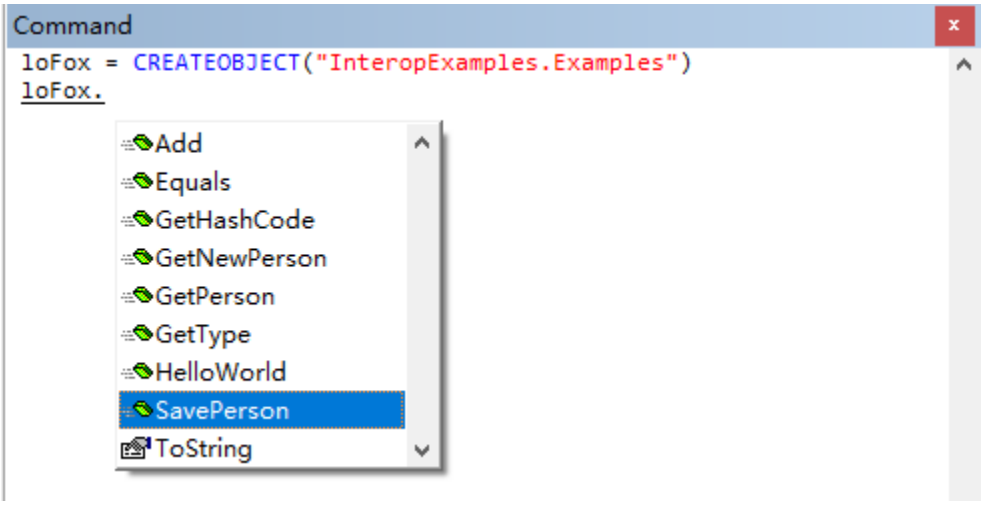


图 4 – 原生COM 互操作对象不显示 persons 列表属性，因为它是泛型类型。

因为类型是泛型 COM 互操作，它是不能发布的类型。所以它仍然存在，您仍旧可以访问它，但它并不像您所期望的那样工作：

```
? loFox.Persons && (Object)
? loFox.Persons.Count && Error
```

简而言之，访问loFox.Persons对象是不可能的。 解决此问题的方法是创建另一个方法，将该列表作为可由FoxPro使用的数组返回。

```
public Person[] GetPersons()
{
    return Persons.ToArray();
}
```

在 FoxPro 中你可以使用这样的代码：

```
loPersons = loFox.GetPersons()
? loPersons[1].FirstName && Rick
```

虽然这有效，但还有其他一些问题。 首先，它要求您控制您正在访问的 .NET 程序集 - 如果您正在处理 CLR 或第三方组件，则无法轻松地向该组件添加方法。 一种选择是创建一个提供这种功能的 .NET 包装类，但这可能很乏味，因为除了泛型之外，还有很多其他的东西。

另一个问题是 COM 互操作会将数组作为一个 FoxPro 数组来处理，而不是原始 .NET 数组。.NET 数组具有 .Length 属性和一组方法，使您可以操

作数组，但在这种情况下，这些功能都会丢失。你当然可以使用函数（如 ALEN()）来确定数组的大小并重新定义数组，但对数组的任何操作都是本地的。实际上，这意味着数组是原始副本，因为一旦进入到 FoxPro ，数据就无法更新原生数组并将这些值反映在 .NET 中。数组是通过值有效传递的。

原生 COM 互操作的缺点

即使在这个非常简单的例子中，我们也已经看到了一些原生 .NET COM 互操作的短处。 最大的问题在于'官方' .NET COM 引擎非常保守地访问 .NET 对象，并执行大量的转换 - 至少在 Foxpro 的情况下 - 导致对象丢失其原始的 .NET 类型信息。

类型访问问题

这是原生COM 互操作常见的问题。 它不能处理很多.NET类型的结构。 以下是原生COM 互操作无法访问的一些内容列表：

- 泛型类型
- 结构和值类型超出了基本内置 .NET
- 枚举
- 静态方法和属性
- 集合和字典
- 二进制数据
- GUIDs

- 数字转换（数字始终为双精度转换）

这只是一些不起作用的高级对象类型的示例。 还有很多其他场景。

数组控制

FoxPro 无法直接处理 .NET 的许多集合类型，泛型或原始 IEnumerable 列表。 在某些情况下，它不起作用，而在另一些情况下，语法很难发现，即使它的工作数组也被编组到 FoxPro 数组中，这些数组通过对数组数据的引用更新几乎不可能。

COM 注册

为了使 .NET COM 对象可访问，必须使用名为 RegAsm 的自定义工具进行注册。 虽然 COM 注册使用熟悉的 CreateObject 提供了很好且简单的语法，但注册一个需要管理员权限的特殊工具仍然很痛苦。 很少有安装程序支持迄今为止的RegAsm注册（因为 .NET COM 互操作并不是非常普遍的边缘情况），所以它通常由应用程序本身来处理注册过程。 如果您有兴趣，我已经在过去的文章中介绍了这个主题。

wwDotnetBridge 的补救措施

wwDotnetBridge为许多这些情况提供了解脱。 它提供了一个备用的 .NET Runtime 托管环境和一个功能强大的 .NET 代理类，它允许您

从 .NET 运行时的上下文中访问和执行 .NET 的大部分功能，避免了一些 COM 封送处理问题，这是原生 COM 互操作的主要缺点。 此代理机制将大部分 .NET 打开到您的 FoxPro 应用程序，其中原生 COM 互操作仅允许您访问启用了 COM 的少数组件。

以下是wwDotnetBridge提供的一些内容:

不需要 COM 注册

也许使用 wwDotnetBridge 的最好理由之一是能够访问未明确导出到 COM 的 .NET 组件。 您可以直接访问 .NET Framework 本身，第三方库或甚至您自己的 .NET 组件中的任何 .NET 组件，而无需任何 COM 注册要求。这意味着如果您需要从您的 FoxPro 代码调用.NET组件，则可以在安装过程中将它们与您的应用程序一起复制，并通过简单的 xCopy 部署直接从磁盘访问它们。 不需要COM注册!

为了清楚，wwDotnetBridge 仍然使用 COM 互操作，但它使用了一种不同的运行时托管机制，它提供了一个 .NET 内部的钩子来实例化和向 FoxPro 传递对象实例。 通过使用 .NET 代理加载新的 .NET 对象实例，COM 实例化过程和注册要求已被删除。

激活的代码和 FoxPro 有点不同 - 您无需使用CREATEOBJECT()，而是在wwDotnetBridge 的库中加载函数来加载依赖程序集（.NET dll）并实例化实际的.NET类:

```
do wwDotNetBridge && Load library
loBridge = CreateObject("wwDotNetBridge","V4")
loBridge.LoadAssembly("bin\InteropExamples.dll")
loFox = loBridge.CreateInstance("InteropExamples.Examples")
```

```
loPerson = loFox.GetNewPerson()
...
```

只有在需要加载显式程序集时才需要 LoadAssembly。 某些核心 .NET 库（System，System.Data，System.Web，System.Web.Services）会自动加载，因此如果您访问 .NET Base Class Library（BCL）功能，则无需显式加载它们。 如果您加载外部程序集并且它们具有依赖关系，则.NET会在加载请求的程序集时自动加载相关程序集。 只要确保所有相关的程序集与加载的程序集都位于相同的文件夹中即可。

wwDotnetBridge 默认使用 .NET 2.0，如果要加载 .NET 4.0（或 4.5，如果安装了4.5），则使用"V4"作为CREATEOBJECT() 调用的参数。 只能运行一个版本的运行时，并首先调用 wins。

请注意，wwDotnetBridge 可以重复使用，通常您会创建一次，然后将其缓存在全局应用程序属性或公共变量中。

使用参数化构造函数创建对象

loBridge.CreateInstance() 还支持实例化具有参数化构造函数的类型，这是 .NET 中的常见要求，而原生 COM 互操作则不可能。 您只需在类型名称后面传递参数值即可访问 .NET 对象的非默认构造函数。

```
loEventLog = loBridge.Createinstance("System.Diagnostics.EventLog",;
                                     "Application",".",;
                                      lcSource)
```

对静态方法和属性的访问

静态方法中包含许多有用的原生 .NET 运行时功能。 静态方法（和属性）与 FoxPro 全局函数类似，意味着它们在没有显式类型实例的情况下被调用。本地 COM 互操作无法访问任何静态内容。使用 wwDotnetBridge，你可以很容易地使用这样的代码：

```
? loBridge.InvokeStaticMethod(
    "System.Net.NetworkInformation.NetworkInterface",;
    "GetIsNetworkAvailable")
```

调用静态方法。 这里没有参数传递，但可以在方法名称后传递任何参数。还有一些 GetStaticProperty() 和 SetStaticProperty() 方法。 静态支持还允许访问 .NET Enumerables（实质上是Enum类型的静态成员），这是许多 .NET API 的另一个经常需要的功能。

支持有问题的类型

我们在之前的简单示例中看到过一种问题类型 - 一种无法通过原生 COM 互操作访问的泛型类型。 wwDotnetBridge 提供了一系列间接执行代理方法，可以动态地在.NET框架内执行代码。 而不是通过调用方法或通过 COM 访问属性的 FoxPro 对象，wwDotnetBridge **在 .NET 运行时本身内**执行 .NET 命令。 这允许访问很多功能，这些功能根本无法通过简单的 COM 互操作工作，因为值和成员调用从未实际通过 COM。

这在wwDotnetBridge中用一些强大的间接执行方法完成：

- InvokeMethod()

- GetProperty()/GetPropertyEx()
- SetProperty()/SetPropertyEx()
- InvokeStaticMethod()
- GetStaticProperty()
- SetStaticProperty()

这些方法全部在 .NET 内部运行，允许访问结构，值类型，枚举，泛型，Guid，二进制数据等等。 此外，这些方法知道FoxPro无法处理的几种类型并自动转换它们。例如，Guids 是一种值类型，不能通过 COM 传递（即使使用 wwDotnetBridge）。相反，wwDotnetBridge 会创建一个包装原始 GUID 的自定义 ComGuid 对象。所有 COM Nulls 都作为 DbNull 对象传递 – wwDotnetBridge 自动将 DbNull 转换为空值。FoxPro二进制值不直接映射到 byte []，但wwDotnetBridge会自动修复二进制数据。还有更多的自动修正功能。

数组和集合处理

许多 .NET API 在数组和集合中保存值。传递给 FoxPro 的集合存在很多常见问题。我们已经看到了一个常见的问题，那就是集合中经常使用通过 COM 互操作不直接支持的泛型（比如List <T>）但是，使用间接引用可以访问我们以前无法访问的Persons集合：

```
loPerson = loBridge.GetPropertyEx(loFox,"Persons[0]")
? loPerson.FirstName
```

GetPropertyEx() 允许提供属性的名称作为对象层次结构，或者在这种情况下为数组或集合索引器。这样工作就会正常。

另一种选择是使用内置于 wwDotnetBridge 中的 ComArray（Westwind.WebConnection.ComArray .NET类型）功能。 ComArray 是一个实际 .NET 数组的封装，并将该数组留在 .NET 中。然后使用 ComArray 的方法来操作数组 - 添加，删除，更新元素，创建新项目，清除等等。ComArray 的关键特性是 .NET 数组永远不会封送到 FoxPro，因此，您对添加的数组元素或元素所做的任何更改都会立即反映在仍然存在于 .NET 中的实时 .NET 数组实例中。您可以将 ComArray 实例传递回 .NET，以代替将该数组作为 InvokeMethod() 或 SetProperty() 调用的参数。

ComArray 的另一个特性是可以将 IEnumerable 类型转换为数组。在 Persons 示例中，Persons 属性实际上是 List <T> 类型，FoxPro 无法访问它，因为通过 COM 互操作期间不支持泛型类型。列表<T> - 与所有数组，集合和字典一样 - 实现 IEnumerable。可以使用 ComArray 将不支持的通用列表转换为具有以下 ComArray 代码的数组：

```
*** Convert List<T> into an plain array
loPersonArray= loBridge.CreateArray()
loPersonArray.FromEnumerable(loFox.Persons)
FOR lnX = 0 TO loPersonArray.Count-1
    loPerson = loPersonArray.Item(lnX)
ENDFOR
```

瞧 - 我们现在可以访问FoxPro中的泛型类型。同样的方法可以用于许多 IEnumerable 实现。IEnumerable 返回的一个常见用例是 LINQ，它始终返回不能直接通过 COM 传递的 IEnumerable <T> 结果。 使用 ComArray.FromEnumerable()，可以将LINQ结果转换为数组并访问。

自动 ComArray 转换

当使用InvokeMethod，GetProperty等间接wwDotnetBridge方法时，SetProperty数组将自动转换为COM Array或从COM Array转换。 如果将一个FoxPro数组传递给.NET，则wwDotnetBridge将从FoxPro数组创建一个ComArray，并将各个项添加到该数组中。 当一个结果从.NET返回时，结果将是一个ComArray对象。

因此，如果我现在使用loBridge.InvokeMethod() 调用GetPersonArray() 方法，结果会自动转换为 ComArray：

```
*** Result is Person[] - automatically returned as ComArray
loPersonArray = loBridge.Invokemethod(loFox,"GetPersons")
FOR lnX = 0 TO loPersonArray.Count-1
    loPerson = loPersonArray.Item(lnX)
    ? loPerson.FirstName + " " + loPerson.Address.Street
ENDFOR
```

请注意，间接方法功能强大，但可选。您仍然可以像使用普通的 COM 互操作一样使用直接的COM 属性/方法访问 - 当直接访问不起作用时您只需要使用间接访问方法，或者您需要额外修复它们提供的功能。

这些修复功能非常强大，可以访问大量功能，否则这些功能无法使用。

使用 wwDotnetBridge 查看原始示例

为了说明我所描述的内容，让我们看看我们的原始示例，使用 wwDotnetBridge 重做。 我们不必在 .NET 代码中改变任何东西 - 代码将按原样运行。 但是，如果我们使用 wwDotnetBridge，我们可以删除我们在编译

时设置的自动 COM 注册 - 让我们这样做来验证我们可以在没有 COM 注册的情况下实例化 .NET 组件。

要禁用 COM 注册，请返回到图3所示的"生成属性"，并取消选中"注册COM互操作"复选框。 确保你退出 VFP，然后重新编译你的项目。

现在在 VFP 中尝试使用以下方法调用 COM 对象：

```
loFox = CREATEOBJECT("InteropExamples.Examples")
```

这应该不再适用 - .NET类不再向COM注册了。

现在让我们从早先重写Persons示例代码来使用wwDotnetBridge。

FoxPro – 用wwDotnetBridge运行原始示例

```
do wwDotNetBridge && 载入库
LOCAL loBridge as wwDotNetBridge
loBridge = CreateObject("wwDotNetBridge","V4")
*** 加载我们的自定义程序集并检查错误
IF !loBridge.LoadAssembly("InteropExamples.dll")
    ? loBridge.cErrorMsg
ENDIF
loFox = loBridge.CreateInstance("InteropExamples.Examples")
IF loBridge.lError
    ? loBridge.cErrorMsg
ENDIF
*** 此代码和原生 COM 代码相同
loPerson = loFox.GetNewPerson()
loPerson.Id = 1
loPerson.FirstName = "Rick"
loPerson.LastName = "Strahl"
loPerson.Address.Street = "32 Kaiea Place"
loPerson.Entered = DATETIME()
loFox.SavePerson(loPerson)
loPerson = loFox.GetNewPerson()
loPerson.Id = 2
loPerson.FirstName = "Markus"
loPerson.LastName = "Egger"
loPerson.Address.Street = "213 Mud Lane"
loPerson.Entered = DATETIME()
loFox.SavePerson(loPerson)
loPerson = null
*** 间接引用允许直接访问常规列表
loPerson = loBridge.GetPropertyEx(loFox,"Persons[0]")
? loPerson.FirstName + " " + loPerson.Address.Street
*** 更好的方式：转换 List<T> 为一个普通的 ComArray
loPersonArray= loBridge.CreateArray()
loPersonArray.FromEnumerable(loFox.Persons)
FOR lnX = 0 TO loPersonArray.Count-1
    loPerson = loPersonArray.Item(lnX)
    ? loPerson.FirstName + " " + loPerson.Address.Street
ENDFOR
```

代码从加载 wwDotnetBridge 库开始。它是一个单独的 PRG 文件加上包含 wwDotnetBridge 的两个DLL文件。

接下来你创建一个实例：

```
loBridge = CreateObject("wwDotNetBridge","V4")
```

默认情况下，wwDotnetBridge加载.NET运行时版本2.0。 您可以使用“V4”或初始化参数的完整版本号加载4.0。 版本号很重要，一次只能加载一个版本。 这是为什么我建议您只在启动时创建一次wwDotnetBridge实例，然后将其全局缓存以便在您的应用程序中进行共享访问的原因之一，以确保您始终使用相同的加载机制。

接下来，我加载包含示例的InteropExamples程序集。 如果您正在访问基本库中的本机.NET功能，则不需要加载默认情况下加载的任何程序集。 如有必要，您可以使用DLL的完整路径 - 这里的DLL位于当前路径中。

```
IF !loBridge.LoadAssembly("InteropExamples.dll")
    ? loBridge.cErrorMsg
ENDIF
```

加载程序集时检查错误是一个好主意，这样您就可以知道什么时候失败了。最常见的情况是无法加载所需的依赖关系，或者您正在将V4 DLL加载到V2运行时。切记，在载入时检查错误!

要创建.NET类的实例，请使用具有完全限定的.NET类型名称的 loBridge.CreateInstance() 方法。

```
loFox = loBridge.CreateInstance("InteropExamples.Examples")
```

完全限定名称的表示方式是 *namespace.classname*。

一旦你创建了一个引用，就好像你使用了CREATEOBJECT() 来在原生 COM 互操作中实例化 COM 对象一样，并且你可以运行完全相同的直接访问代

码。wwDotnetBridge 仍然像使用 COM 互操作一样使用 COM 来传递对象，但是使用 loBridge 实例中的间接调用功能可以获得额外的功能。例如，要访问Persons集合（这是不支持的泛型类型），可以使用以下语法：

```
loPerson = loBridge.GetPropertyEx(loFox,"Persons[0]")
? loPerson.FirstName + " " + loPerson.Address.Street
```

使用间接访问函数，您总是传递对基础对象的引用，以及要访问的成员的字符串。 非Ex方法需要访问确切的成员名称（“FirstName”，“StreetName”，“Persons”）。 Ex版本允许访问嵌套属性（“Address.Street”）和数字索引器（“Persons [0]”）。

最后，在此调用中发生自动结果值和参数修正：

```
*** 结果为 Person[] - 自动作为 ComArray 返回
loPersonArray = loBridge.Invokemethod(loFox,"GetPersons")
For lnX = 0 To loPersonArray.Count-1
    loPerson = loPersonArray.Item(lnX)
    ? loPerson.FirstName + " " + loPerson.Address.Street
Endfor
```

GetPersons 方法返回 .NET 中的 Person [] 结果，Invokemethod 调用将此结果修复为 ComArray。 因此，您可以使用 Item（x），Add(element)，Remove(x)，Clear() 和 Count 属性等方法来轻松操作数组元素并添加新元素。 还有一个有用的 CreateItem() 方法，可用于创建新的 .NET 元素并将其传回给 FoxPro，以便您可以轻松地为大多数数组创建新元素。

wwDotnetBridge 是如何工作的

wwDotnetbridge与一些互操作组件一起工作：

- ClrLoader.dll – .NET Rruntime 的 Win32 载入器
- wwDotnetBridge.dll – 帮助访问和调用 .NET 成员的 .NET 代理
- wwDotnetBridge.prg – .NET 代理类的 FoxPro 前端类

图5显示了wwDotnetBridge的结构。

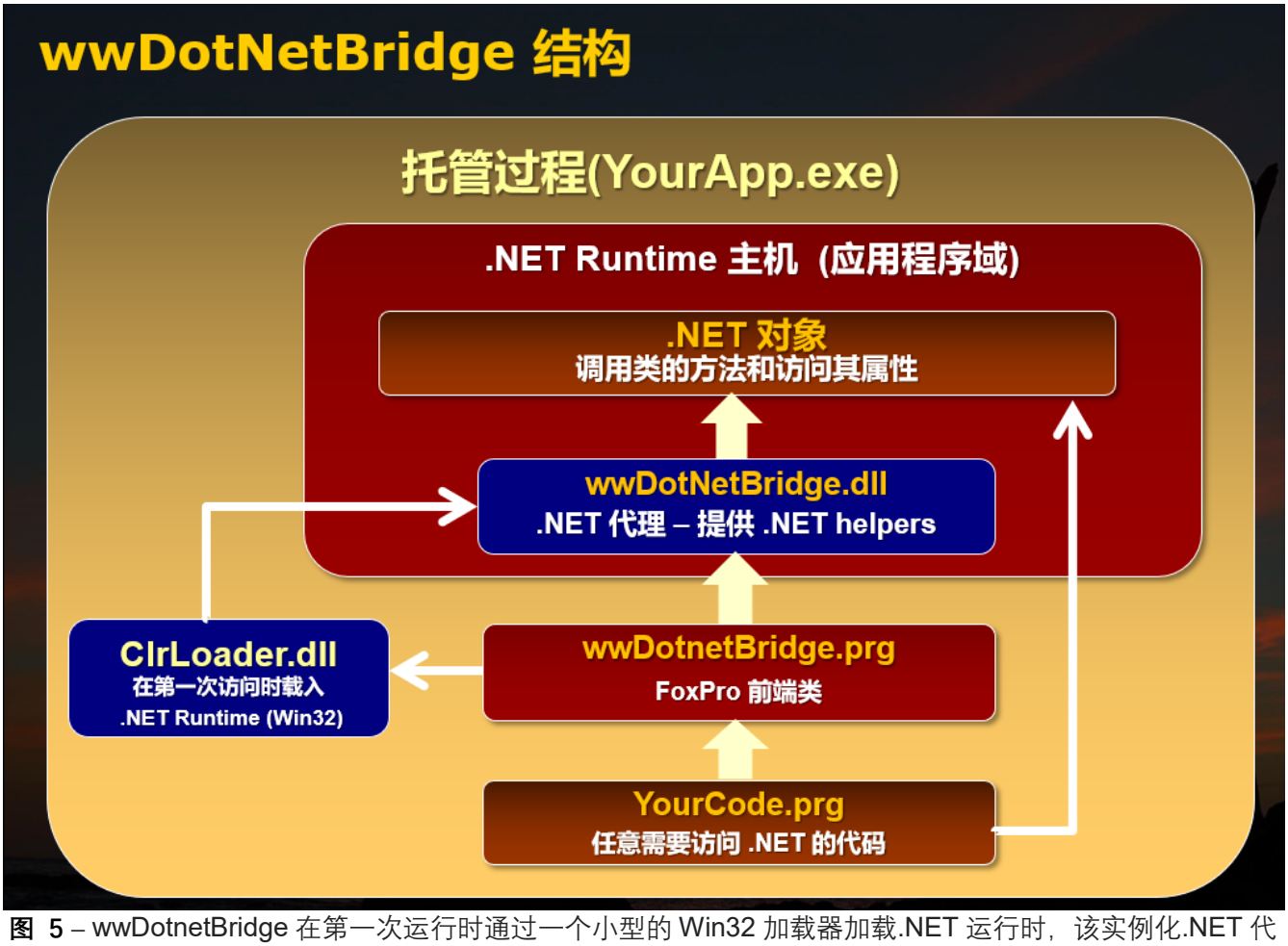


图 5 – wwDotnetBridge 在第一次运行时通过一个小型的 Win32 加载器加载.NET 运行时，该实例化.NET 代理并将其传回给 FoxPro。.

当您使用wwDotnetBridge并首先创建它的一个实例时，该库加载您在启动参数中指定的.NET Runtime:

```
loBridge = CreateObject("wwDotNetBridge","V4")
```

这通过一个小型 ClrLoader Win32 DLL（或商业 West Wind 工具中的wwIPstuff.dll）完成。 ClrLoader 创建.NET运行时的实例，加载

wwDotnetBridge.dll 并创建 .NET wwDotnetBridge 代理类的实例。 该代理然后返回到 FoxPro 作为 COM 引用使用原始 COM 接口指针。

FoxPro – wwDotnetBridge::Load 方法载入 .NET runtime

```
Function Load()

If Vartype(This.oDotNetBridge) != "O"
    This.SetClrVersion(This.cClrVersion)

    If This.lUseCom
        This.oDotNetBridge = Createobject("Westwind.wwDotNetBridge")
    Else
        *** 通过文件名载入 - 假定 wwDotNetBridge.dll 在搜索路径中
        Declare Integer ClrCreateInstanceFrom In WWC_CLR_HOSTDLL
        String, String, String@, Integer@

        lcError = Space(2048)
        lnSize = 0
        lnDispHandle = ClrCreateInstanceFrom(
        Fullpath("wwDotNetBridge.dll"),;
            "Westwind.WebConnection.wwDotNetBridge", @lcError,@lnSize)

        If lnDispHandle < 1
            This.SetError( "Unable to load Clr Instance. " + ; Left(lcError,lnSize) )
            Return Null
        Endif

        *** 转换句柄为 IDispatch 对象
        This.oDotNetBridge = Sys(3096, lnDispHandle)

        *** 这里显式的添加引用 -
        *** 否则当对象释放时会发生怪异的事情
        Sys(3097, This.oDotNetBridge)
        If Isnull(This.oDotNetBridge)
            This.SetError("Can't access CLR COM reference.")
            Return Null
        Endif
    Endif

    This.oDotNetBridge.LoadAssembly("System")
Endif
Return This.oDotNetBridge
```

.NET代理的 *this.oDotnetBridge* 实例用于 FoxPro wwDotnetBridge 类在内部与您使用类的方法进行交互。只有在实例化对象时才调用Load。 如果您在您的应用程序中多次调用此代码，C ++加载器代码会尝试重新加载 .NET 运行时，但内部 .NET API 会检测到运行时已在运行，并简单地将 wwDotnetBridge .NET 组件加载到该运行时中，指向 FoxPro 的指针。 虽然这很快，但不要反复调用这些代码是一个好主意，因为它有一些开销。 这也是为

什么缓存 wwDotnetBridge 实例是个好主意的原因之一。

一旦加载运行时，ClrLoader不再使用 - 它仅用于以 wwDotnetBridge :: oDotnetBridge 属性的形式将 FoxPro wwDotnetBridge 类检索到 .NET 代理实例，然后在内部使用它来将调用转发给.NET。

wwDotnetBridge .NET 代理包含大量提供间接对象访问的方法，然后将这些方法映射到Visual FoxPro wwDotnetBridge类。代理类有两个主要目的：

- **它是 .NET 实例的工厂**

  如果没有.NET wwDotnetBridge.CreateInstance() 方法，则无法加载新的 .NET 类。

- **它是 .NET 内部的代理**

  它就像一个进入.NET内部的窗口。 您可以调用方法并将内容存储在其他属性或 ComValue 实例上，这样您就可以将代码保存在 .NET 内部，而无需 COM 封送处理，从而提供对许多功能的访问，否则这些功能无法通过COM进行工作。 FWIW，代理功能也可以使用原生 COM 互操作来实现，而 wwDotnetBridge 实际上可以使用原生 COM 互操作实例化并与本地调用的（CREATEOBJECT()）COM对象进行交互。

FoxPro类拥有对.NET代理类的引用，并且基本上可以传递对.NET代理的调用。 所以当你调用时：

```
loPersons = loBridge.GetProperty(loFox,"Persons")
```

FoxPro代码在内部执行：

```
Function GetProperty(loInstance,lcProperty)
Return This.oDotNetBridge.GetProperty(loInstance, lcProperty)
Endfunc
```

像 InvokeMethod 这样的一些方法在处理多个参数方面稍微复杂一些，但总的来说，FoxPro 类主要是将调用转发给 .NET 代理，并返回结果和一些参数修正，以稍微处理重载比 .NET 调用更简洁。理论上讲，也可以直接使用 loBridge.oDotnetBridge 属性调用 .NET 方法，但 FoxPro 包装器为 FoxPro 开发人员提供了一个更清晰的接口。 对于 FoxPro 开发人员来说，这是一个简单的单一类接口来进行交互。

wwDotnetBridge的完整源代码 - Win32 DLL，.NET Proxy和FoxPro类可以从GitHub下载并从中查看：

GitHub 上的 wwDotnetBridge 项目（英文）

GitHub 上的 wwDotnetBridge 项目（中文）

wwDotnetBridge 示例

OK,现在你已经对 wwDotnetBridge 如何工作有了了解。让我们来看一些例子，看看你在实践中能做什么。

我们来看一个在内置.NET框架函数中调用静态方法的示例。 使用普通的 COM Interop不可能做到这一点，因为静态方法不是从类型引用中调用，而是直接作为静态函数来访问。

以下内容会收到机器网络连接的状态：

```
loBridge = CreateObject("wwDotNetBridge","V4")
? loBridge.InvokeStaticMethod( "System.Net.NetworkInformation.NetworkInterface",;
                               "GetIsNetworkAvailable")
```

如果您连接到网络，则此方法调用返回true。 如果您断开网络电缆或关闭无线适配器，它将返回错误。 超级简单，但非常有用，特别适用于可能是移动设备的应用程序，并且需要先检查连接性，然后再下载数据。

InvokeStaticMethod() 通过提供包含静态方法的完整.NET类型工作 - 在本例中为 System.Net.NetworkInformation.NetworkInterface。您可以将类和方法指定为字符串值，然后指定任何附加参数值（如果方法接受参数 - 这个例子不接受参数）。

静态方法很常见，尤其是基础 .NET 库中的系统功能。我们来看另一个例子，它允许您写入Windows事件日志 - 这是一个使用 Windows API 的非常复杂的过程。 使用 .NET，代码会更加简单，使用 wwDotnetBridge 可以从 FoxPro 访问 .NET 组件。

FoxPro – 在 Windows 事件日志中书写两条记录

```
loBridge = CreateObject("wwDotNetBridge","V4")
lcSource = "FoxProEvents"
lcLogType = "Application"

IF !loBridge.Invokestaticmethod("System.Diagnostics.EventLog",;
                                "SourceExists",lcSource)
    loBridge.Invokestaticmethod("System.Diagnostics.EventLog",;
                                "CreateEventSource",;
                                Source,lcLogType)
ENDIF

*** 书写默认消息 - 信息
* public static void WriteEntry(string source, string message)
loBridge.Invokestaticmethod("System.Diagnostics.EventLog",;
                            "WriteEntry",lcSource,;
                            "Logging from FoxPro " + TRANSFORM(DATETIME()) )

*** 使用 COM Value 来投射枚举类型
loValue = loBridge.CreateComValue()
loValue.SetEnum("System.Diagnostics.EventLogEntryType.Error")

* public static void WriteEntry(string source, string message, EventLogEntryType type, int
eventID)
loBridge.Invokestaticmethod("System.Diagnostics.EventLog",;
                            "WriteEntry",;
                            lcSource,;
```

```
"Logging error from FoxPro " +
TRANSFORM(DATETIME()),;
loValue, 10 )
```

这是使用一堆静态方法访问 .NET 的系统功能的另一个示例。 第一种方法检查 SourceExits() 函数是否存在事件源。事件日志由一个源创建，该源通过名称和要写入的日志类型（应用程序，系统，安全性等）来标识日志。 进程使用 CreateEventSource() 创建事件源，然后使用 WriteEntry() 写入它。 一旦事件源存在，您可以调用 WriteEntry() 写入源。 这将一个默认类型的条目写入事件日志 - 这是一个信息条目。

第二个日志条目稍微复杂一些，因为我指定了错误的 EventLogEntryType 而不是默认值。这是需要传递的 Enum 值的。wwDotnetBridge 包含一些简单的函数来用 GetEnumValue 检索 Enum 值：

```
leValue = loBridge.GetEnumValue("System.Diagnostics.EventLogEntryType.Error")
? leEvalue && 1
```

在大多数情况下，这个更简单的函数用于获取枚举值，然后可以将它传递给任何需要枚举值的.NET方法。 然而在这里它不起作用，因为WriteEntry接受许多重载，并且GetEnumValue（）返回的整数值没有正确映射到方法重载。

ComValue 提供“真正的” .NET 值

wwDotnetBridge 包含一个 .NET ComValue 类型（Westwind.WebConnection.ComValue），它有一个value属性。ComValue 上的方法可以直接从 .NET 中加载值结构，而无需先将值封送到 FoxPro。 对于我们的枚举值，这意味着我可以创建一个枚举值并直接将其存储在值结构中，如下所示：

```
LOCAL loValue as Westwind.WebConnection.ComValue
```

```
loValue = loBridge.CreateComValue()
loValue.SetEnum("System.Diagnostics.EventLogEntryType.Error")
```

现在，当使用 InvokeStaticMethod() 而不是传递枚举时，我传递 ComValue() 实例，并且 wwDotnetBridge 自动为此获取 loValue.Value 属性，并将其用作参数：

```
loBridge.Invokestaticmethod("System.Diagnostics.EventLog",;
                            "WriteEntry",;
                            lcSource,;
                            "Logging error from FoxPro " +
                            TRANSFORM(DATETIME()),;
                            loValue, 10 )
```

这听起来有点复杂，但是，幸运的是这种情况比较少，此时你不得不求助于 ComValue。你可以将它作为最后一招。它展示了 wwDotnetBridge 的一些功能，以避免将值传入 FoxPro 以避免 COM 类型转换。ComValue 上的其他方法可以从 SetValueFromProperty()，SetValueFromInvokeMethod() 和 SetValueFromStaticProperty() 中设置值，以及从 FoxPro值创建 .NET 类型的许多类型特定的加载器。

毕竟我们现在已经在Windows事件日志中写入了两个条目。

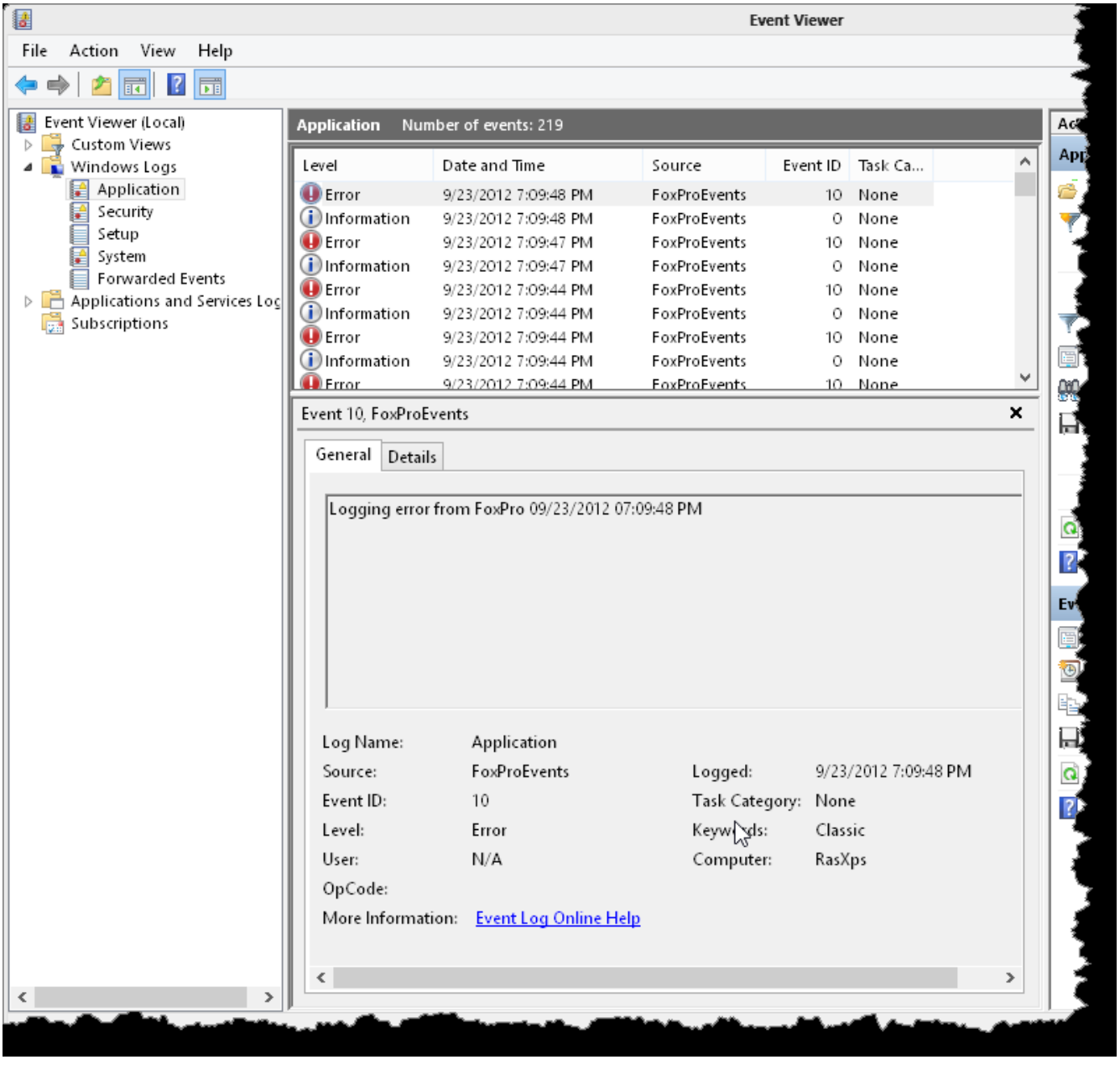


图 6 – *您可以看到从我们的FoxPro代码写入的事件。 第一个条目是错误日志类型并设置了事件ID，第二个条目仅包含消息和默认图标。*

为了完成这个事件日志主题，我们来看看如何列出事件日志项目：

```
*** 显示事件日志条目
loEventLog = loBridge.CreateInstance("System.Diagnostics.EventLog")
loEventLog.Source = lcSource
loEventLog.Log = "Application"

*** 转换日志条目为一个 ComArray 类
*** 间接访问会自动将 .NET 数组转换为 ComArray
loEvents = loBridge.GetProperty(loEventLog,"Entries")
? "Entries: " + Transform(loEvents.Count)
```

```
lnTo = Min(loEvents.Count,10)
For lnX = loEvents.Count-1 To loEvents.Count-lnTo Step -1
    loEvent = loEvents.Item(lnX)
    ? loEvent.Message
    ?
Endfor
```

这里的 EventLog .NET 类型是作为一个实例创建的，我们通过它的属性指定了源和日志。请注意，我可以直接使用这些值。然而，Entries 属性是一个数组，为了更容易在 FoxPro 中使用它并保持 .NET 中的数组，我可以调用 GetProperty() 来检索数组作为 ComArray 实例。 通过这样做，我获得了使用 Item() 方法轻松操作数组并遍历数组的能力。

查找 .NET 类型签名

在这一点上，你可能会认为这很好，很整齐，但你怎么知道你怎么找到他们呢?

幸运的是，你不需要依赖文档。 有许多工具可以用来浏览.NET程序集并查看它们包含的类和成员。 以下是其中一些工具:

- Red Gate's .NET Reflector (Ver7 之前的版本时免费的)
- IL Spy (开源)
- JetBrain's DotPeek (免费)
- Telerik's JustCode (带有广告)

就我个人而言，我很喜欢Reflector，但其中的任何一个都可以实现（Reflector 6.5 包含在示例的Tools文件夹中），所以我在这里使用它（它同样包含在Tools文件夹中的示例中）。

例如，要查看 EventLog 的功能，我们可以看看 System.dll。 在

Reflector 中，核心 .NET 框架程序集通常默认加载，我可以导航到 System.dll 和 System.Diagnostics 命名空间，然后导航到 EventLog 类。

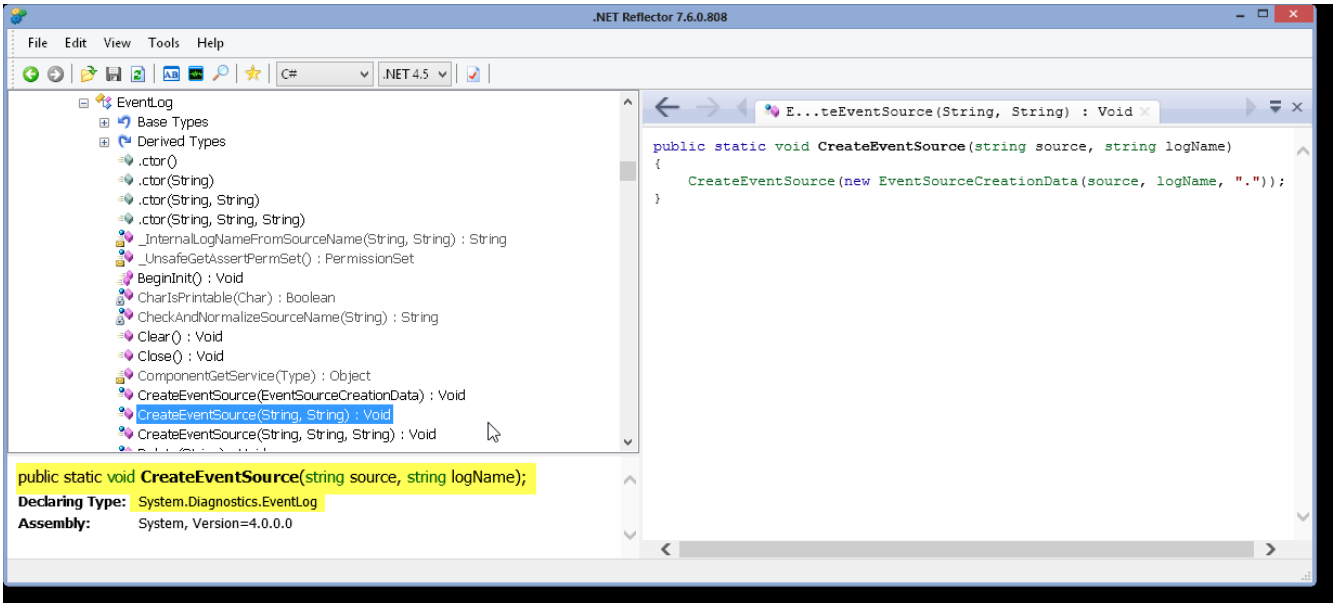


图 7 – *Reflector 是浏览.NET Framework库或任何.NET程序集的好工具，并查看可用的功能。 大多数代码也可以在反编译模式下查看，除非明确地加以混淆。*

Reflector 可以轻松浏览.NET程序集并查找您正在查找的类，方法或成员。您通常需要知道的调用.NET类型的重要内容是：

- 要创建的类型的名称（或要访问的静态实例名称）
- 方法调用的确切参数签名

在图5中，您可以看到为CreateEventSource方法突出显示的所有内容，这些方法在通过wwDotnetBridge调用时看起来像这样：

```
loBridge.Invokestaticmethod("System.Diagnostics.EventLog",;
                            "CreateEventSource",;
                            Source,lcLogType)
```

调用第三方组件

与 .NET 集成的一个很好的理由是从 FoxPro 代码调用第三方功

能。 .NET在开源组件和商业产品方面都有很多功能，您可以使用 wwDotnetBridge 轻松地从 FoxPro 中轻松使用此功能。 该过程几乎与我上面展示的相同，只是必须明确加载 .NET 程序集。

无论是您自己创建的 .NET 程序集，还是来自某个开源项目，您都可以连接并访问外部.NET代码。

以下是另一个示例：我将使用 OpenPop .NET POP3 库访问 POP3 邮箱并检索电子邮件列表。 正如我们对 EventLog 类所做的那样，我们可以通过 Reflector 窥探 OpenPop 的 API。 打开反射器并加载 Openpop.dll。

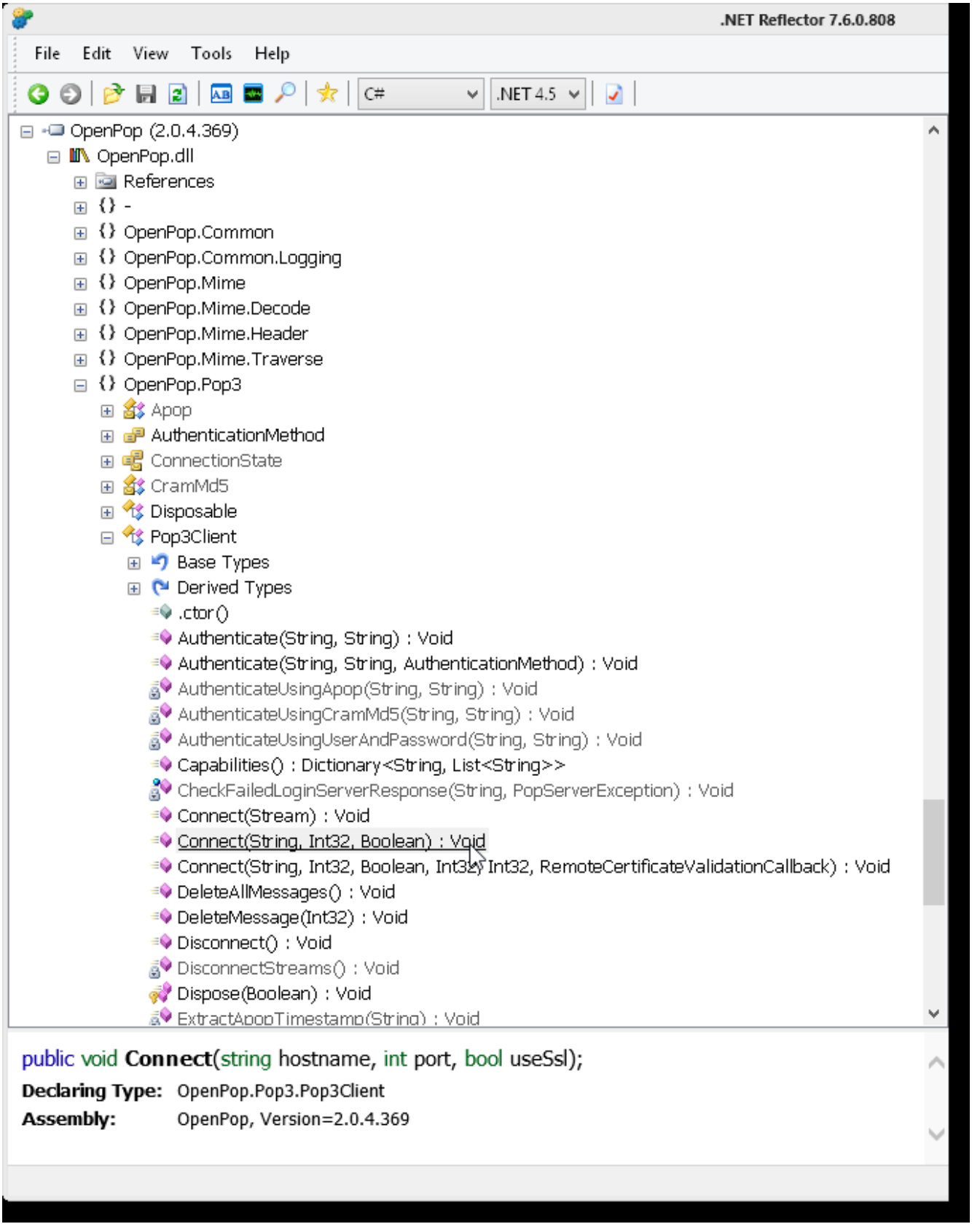

.NET Reflector 7.6.0.808
File Edit View Tools Help
C#
.NET 4.5
OpenPop (2.0.4.369)
OpenPop.dll
References
-
OpenPop.Common
OpenPop.Common.Logging
OpenPop.Mime
OpenPop.Mime.Decode
OpenPop.Mime.Header
OpenPop.Mime.Traverse
OpenPop.Pop3
Apop
AuthenticationMethod
ConnectionState
CramMd5
Disposable
Pop3Client
Base Types
Derived Types
.ctor()
Authenticate(String, String) : Void
Authenticate(String, String, AuthenticationMethod) : Void
AuthenticateUsingApop(String, String) : Void
AuthenticateUsingCramMd5(String, String) : Void
AuthenticateUsingUserAndPassword(String, String) : Void
Capabilities() : Dictionary<String, List<String>>
CheckFailedLoginServerResponse(String, PopServerException) : Void
Connect(Stream) : Void
Connect(String, Int32, Boolean) : Void
Connect(String, Int32, Boolean, Int32, Int32, RemoteCertificateValidationCallback) : Void
DeleteAllMessages() : Void
DeleteMessage(Int32) : Void
Disconnect() : Void
DisconnectStreams() : Void
Dispose(Boolean) : Void
public void Connect(string hostname, int port, bool useSsl);
Declaring Type: OpenPop.Pop3.Pop3Client
Assembly: OpenPop, Version=2.0.4.369

图 8 – *在 Reflector 中找出OpenPop API。这是一种发现 API 的好方法，可以查看您可以访问的内容以及方法签名*

查看API，我们现在可以使用以下代码循环访问消息：

FoxPro – 使用 OpenPop .NET 库访问 POP3 帐户

```
Local loBridge As wwDotNetBridge

loBridge = Createobject("wwDotNetBridge")

? loBridge.LoadAssembly("bin\OpenPop.dll")

loPop = loBridge.CreateInstance("OpenPop.Pop3.Pop3Client")

*** 连接重载不能直接工作
* loPop.Connect("mail.gorge.net",587,.f.)
? loBridge.InvokeMethod(loPop,"Connect","pop3.gorge.net",110,.F.)
? loPop.Authenticate("rstrahl",Strtran(GetSystemPassword(),"0",""))
lnCount = loPop.GetMessageCount()
? StringFormat("{0} Messages",lnCount)

*** 注意: OpenPop是基于1的，因为pop3是基于1的!
** 显示最后的消息
For lnX = lnCount To 1 Step -1
    loHeader = loPop.GetMessageHeaders(lnx)
    ? loHeader.From.DisplayName
    ? " " + loHeader.Subject
    ?
    If lnX < lnCount - 10
        Exit
    Endif
Endfor

loPop.Disconnect()
```

此代码循环播放POP3收件箱中的最后10封邮件，并显示发件人和邮件主题。

此代码明确必须通过指定它在磁盘上的位置来调用 OpenPop 库上的 LoadAssembly()。提供 DLL 的相对路径或完整路径。您还可以使用全局程序集缓存（GAC）中的全限定程序集名称（也可以在类节点的 Reflector 中找到该程序集）并使用 LoadAssembly() 方法加载程序集。例如，下面是一个 LoadAssembly 调用来加载在 GAC 中注册的 System.Web.Extension 程序集：

```
loBridge.LoadAssembly("System.Web.Extensions, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35")
```

然后，当您导航到类或该类的任何成员时，将使用可在 Reflector 中找到的类的全名创建该实例。

接下来打开连接，我必须在 Pop3Client 的 Connect() 方法上使用 InvokeMethod()。直接调用不起作用，因为该方法重载，并且 .NET 不会将 FoxPro 数值参数看作整数，而是将其视为双精度值。FoxPro 数值通常以双精度传递给 .NET，这可能会导致方法调用失败。使用中间 InvokeMethod() 调用和 Reflection 会导致在 .NET 中发生一些额外的类型转换，从而使调用正常工作。

一旦连接了代码的其余部分，就可以直接访问 OpenPop API，而无需对 wwDotnetBridge 进行间接调用 - COM实例具有简单的参数类型，这些参数类型只能在FoxPro中自行使用。

最佳做法是首先尝试使用直接访问，如果不起作用或不产生预期结果，请使用 wwDotnetBridge 的方法和帮助程序。

为了完整起见，这里有一些额外的 OpenPop 代码来检索单个消息并提取内容。这个过程有点涉及，因为它需要处理不同类型的内容（Html，普通，附件等）。

```
*** 按计数（列表后）查找最后一条消息并显示
loMsg = loPop.GetMessage(lnCount)
loMsg.Headers.Subject

loPart = loMsg.FindFirstHtmlVersion()
If Isnull(loPart)
    loPart =loMsg.FindFirstPlainTextVersion()
Endif
```

```
If !Isnull(loPart)
    ? StringFormat("Is Text: {0}",loPart.IsText)
    ShowHtml( loPart.GetBodyAsText() )
Endif
```

创建你自己的 .NET 封装

有时，直接从 FoxPro 访问 .NET 代码会变得非常繁琐，因为您必须在 FoxPro 和 Reflector 中找出 API，并且不断的试错来查看哪些是有效的，哪些不是。如果 API 简单直接且不复杂，则可以使用 FoxPro，但是如果您正在处理大量复杂结构，则创建一些 .NET 组件的封装然后在 FoxPro 中调用它可能会更容易。

创建 .NET 封装以抽象复杂的 .NET 功能

针对电子邮件，我创建了一个包含 System.Net.SmtpClient 类功能的 .NET wwSmtp 组件。事实上，我使用的 FoxPro wwSmtp 类在我们的 West Wind Web Connction 和 West Wind Internet 和 Client Tools 产品中已经公开了。虽然我可以通过使用 wwDotnetBridge 在 .NET 中调用本地 SmtpClient MailMessage API 来直接实现 wwSmtp 类，但我选择创建一个封装并提供一系列帮助程序的 .NET 类。这有两个好处 - 在 .NET 中开发类（如果您懂一些 .NET）会更容易，因为 Visual Studio 为您提供了完整的 Intellisense，以便了解API如何工作。包装 .NE T类型然后暴露一个接口，FoxPro COM 友好并且抽象接口，使其尽可能容易使用 - 在我的情况下，通过 C ++ 组件模拟 Web Connection 中已提供的现有电子邮件功能。最

后，.NET 组件 - 完成后 - 可以在 FoxPro 和 .NET 中使用。

如果您喜欢，可以在提供的示例中查看 SmtpClient 类的 wwSmtp.cs 包装的源代码，它的体积相当大并且不适合此处展示。

以下是使用 wwDotnetBridge 实际访问该类的代码：

FoxPro – 使用 wwSmtp .NET 类来发送一封邮件

```
loBridge = Createobject("wwDotNetBridge","V4")

loBridge.LoadAssembly("InteropExamples.dll")
loBridge.cErRORMSG

Local loSmtp As Westwind.wwSmtp
loSmtp = loBridge.CreateInstance("Westwind.wwSmtp")

*loSmtp.AddAttachment_3("c:\sailbig.jpg")
loBridge.InvokeMethod(loSmtp,"AddAttachment","c:\sailbig.jpg")
loSmtp.MailServer = "smtp.server.com:587"

loSmtp.UseSsl = .T.
loSmtp.Username = "user"

loSmtp.Password = "secret"
loSmtp.Recipient = "Rick Strahl<rstrahl@west-wind.com>"

loSmtp.SenderEmail = "admin@west-wind.com"
loSmtp.Subject = "Test Message"

TEXT TO loSmtp.Message NOSHOW
<html>
    <head>
        <style>
            body { font-family: Verdana; background: cornsilk; }
        </style>
    </head>
<body>
<p>
Hello Rick,
</p>
<p>
This is a test message from <b>Southwest Fox</b>
</p>
<p>
Enjoy,
</p>
<p>
+++ Rick ---
</p>
</body>
</html>
ENDTEXT
```

```
loSmtp.ContentType = "text/html"

IF (!loSmtp.SendMail())
    ? loSmtp.ErrorMessage
ENDIF

? "Mail sent"
```

正如你可以看到该组件加载了 wwDotnetBridge，但是一旦加载，wwDotnetBridge 的方法就没有太多用处。 这是因为 wwSmtp 对 SmtpClient() 组件进行了抽象并简化了它，并通过简单的属性值和简单的方法调用明确地使其易于访问。

实例上的一个 wwDotnetBridge 调用：

```
*loSmtp.AddAttachment_3("c:\sailbig.jpg")
loBridge.InvokeMethod(loSmtp,"AddAttachment","c:\sailbig.jpg")
```

这是必要的，因为该方法重载，否则它将无法工作。

创建 .NET 封装以简化在 FoxPro 中使用的复杂 .NET API，这是一种以更简单更抽象的方式向 FoxPro 公开复杂 .NET 功能的好方法。尽管 wwDotnetBridge 允许访问大多数 .NET 功能，但是创建大量的 .NET 访问代码并不是最好的想法。 wwDotnetBridge 代码可能很冗长，并且经常需要一些试错才能正确使用 .NET 调用。如果您完全熟悉 .NET，那么通过利用 C# 编译器和 Intellisense 来创建 .NET 代码来访问 .NET 中的复杂功能会容易得多。

COM 互操作的事件处理

wwSmtp 类的另一个特性和我创建它的原因之一是支持异步发送电子邮

件。 您可以发送电子邮件，而无需等待完成：

```
? loSmtp.SendMailAsync()
```

这取代了 Send() 调用。这很好用，.NET wwSmtp 类负责启动一个新的线程，并在 .NET 中异步运行这个操作，就像在.NET中一样简单：

```
public void SendMailAsync()
{
    Thread mailThread = new Thread(this.SendMailRun);
    mailThread.Start();
}
protected void SendMailRun()
{
    // Create an new reference to insure GC doesn't collect
    // the reference from the caller
    wwSmtp Email = this;
    Email.SendMail();
}
```

喔~-您拥有了多线程代码。

多线程代码经常出现事件。事件在 COM 互操作中很棘手，这是 wwDotnetBridge 无法帮助的地方。实际上，如果要处理来自 .NET 组件的事件，则必须将. net 组件注册到 COM 中，以使事件接口可用。COM 事件需要 COM 来接收它们，因此，如果您创建的组件需要事件处理，那么使用 RegAsm 注册 COM 将是不可避免的。

了解. NET 事件和 COM 的第一件事是必须显式设置 COM 事件。如果事件没有显式发布到 COM，则无法连接任意的. NET 事件并让它们处理 COM。这意味着. NET 事件处理在 COM 上几乎仅限于您创建自己并自己发布事件的组件。

要创建处理事件的 COM 组件，需要两件事情：

- 将类链接到事件接口的类声明
- 作为方法公开每个事件的接口定义

让我们来看看wwSmtp是如何工作的。 第一步是类声明，如下所示：

```
[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
[ComSourceInterfaces(typeof(IwwSmtpEvents))]
[ProgId("Westwind.wwSmtp")]
public class wwSmtp : IDisposable
```

事件特定属性是ComSourceInterfaces属性，它描述了将事件暴露给COM的事件接口。接口然后需要声明在接口定义中导出的每个事件。

wwSmtp类包含两个事件声明:

```
public event delSmtpNativeEvent SendComplete;
public event delSmtpNativeEvent SendError;
public delegate void delSmtpNativeEvent(wwSmtp Smtp);
```

委托是定义事件签名的函数定义 - 此事件以 wwSmtp 对象实例作为参数触发。从 .NET 代码调用时，您可以直接挂接这两个事件。

但通过 COM wwSmtp 必须注册并公开将这些事件映射到事件的事件接口。一个接口只是一个声明，而不是一个实现，所以这个类非常简单:

```
[ComVisible(true)]
[InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]
public interface IwwSmtpEvents
{
    [DispId(1)]
    void SendComplete(wwSmtp smtp);
    [DispId(2)]
    void SendError(wwSmtp smtp);
}
```

该类只是简单地将两个事件处理程序映射到wwSmtp类。 当C＃编译器编译此代码时，它会自动启动必要的COM事件接口，以便触发COM事件。

在FoxPro中，您需要通过实现事件接口来捕获这些事件。 这个COM接口必须是OLEPUBLIC并且有效地从COM接口继承。 这就是组件必须注册的原因。

```
Define Class wwSmtpEvents As Session OlePublic
    Implements IwwSmtpEvents In "Westwind.wwSmtp"
    Procedure IwwSmtpEvents_SendError(smtp As VARIANT) As VOID
    ? "Sending message '" + smtp.Subject + "' failed..." +;
        smtp.ErrorMessage
    Endproc
    Procedure IwwSmtpEvents_SendComplete(smtp As VARIANT) As VOID
```

```
    ? "Sending message '" + smtp.Subject + "' complete..."
    Endproc
Enddefine
```

事件类中的每个方法都与 .NET 导出的 .NET COM 接口的方法相匹配。

要将事件接口绑定到 COM 类，请使用将源对象和事件接口绑定在一起的 Foxpro EVENTHANDLER() 函数：

```
loEvents = CREATEOBJECT("wwSmtpEvents")
EVENTHANDLER(loSmtp,loEvents)
? loSmtp.SendMailAsync()
? "Code is done executing..."
```

有了这个代码，再次运行代码。 您现在应该看到邮件请求运行并立即执行打印代码执行到屏幕，之后就是发送邮件发送完成消息或发送失败时出现的错误消息。

通过使用FoxPro对象浏览器并选择.NET COM .TLB文件并在对象浏览器中选择事件接口，然后将接口拖放到FoxPro PRG文件中，可以自动为您创建此接口。 当你这样做时，你会为你自动生成一个空的类。

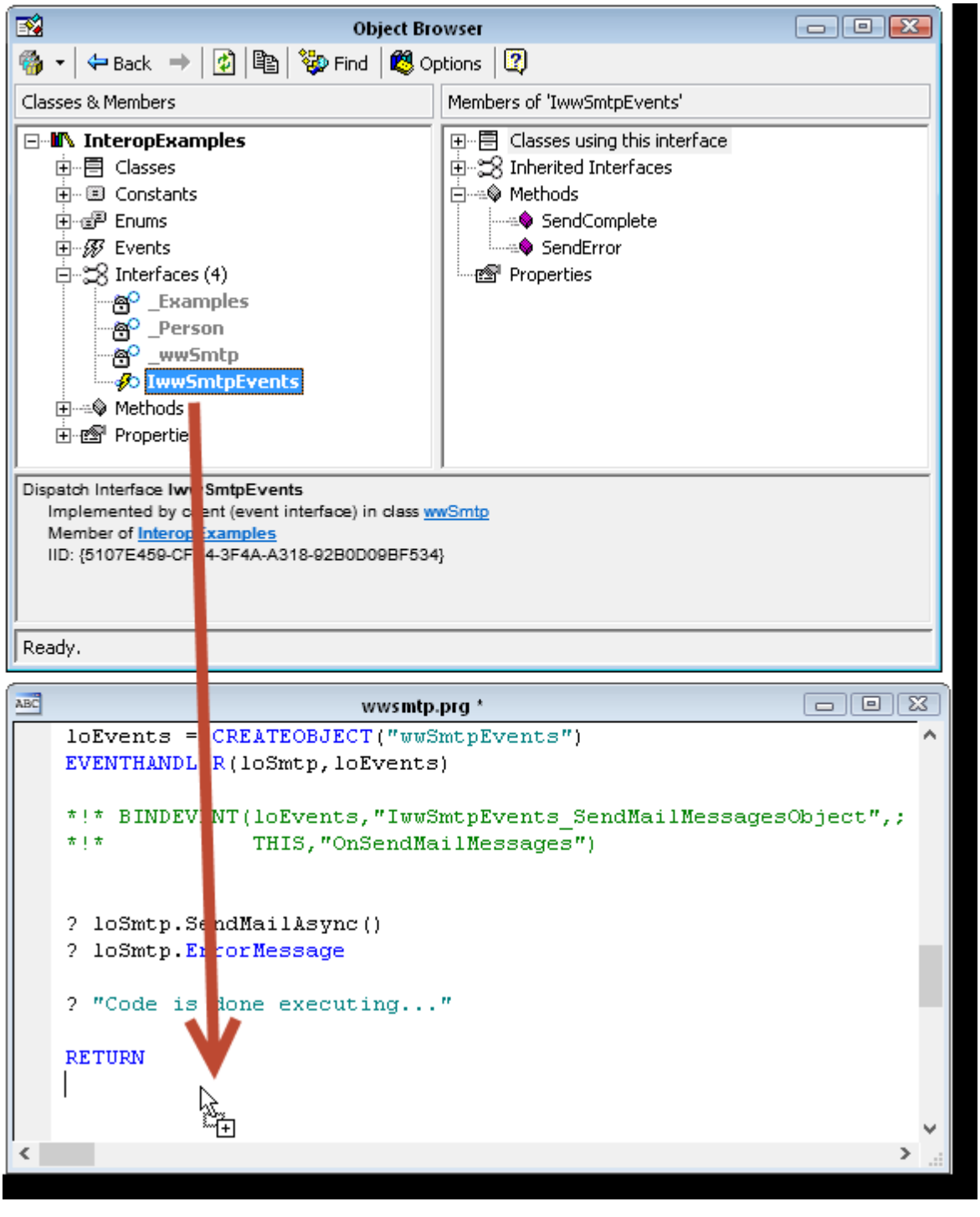


图 9 – *将对象浏览器中的事件接口拖放到FoxPro PRG中将创建COM事件接口。*

wwDotnetBridge 对象修正

你已经看到将值从 .NET 传递给 FoxPro 是相当容易的。 大多数简单的类型，对象，甚至数组几乎可以直接使用。 对于其他一些有问题的类型 ComArray 和 ComValue 可以提供帮助。

我们来看几个特殊类型以及它们如何处理。 让我们从数组开始，因为它们是 COM 互操作中常见的痛苦来源。 让我们回到我们之前使用的 InteropExamples 项目，并向 Examples 类添加一些方法。

数组和 ComArray

当我们查看Persons示例时，我们先简要地看了一下 ComArray 类。 我们有一个 GetPersons() 方法，让我们将 AcceptPersons 方法添加到接受一个 persons 数组作为输入的类。

```
public Person[] GetPersons()
{
    return Persons.ToArray();
}

public bool AcceptPersons(Person[] persons)
{
    Persons.Clear();
    Persons.AddRange(persons);
    return true;
}
```

处理必须通过 COM 传递的数组很困难，因为 COM 封送破坏了原始的 .NET 类型，其结果为 FoxPro 数组。FoxPro 数组不容易被发回到 .NET，因为它失去了它的 DONNET 特性。 但是通过 ComArray 实现，这实际上相

当容易。

让我们在 FfoxPro 中创建一些代码到 GetProcessor()，然后向数组中添加一个新的 person 并将整个数组传回 .NET。

FoxPro – 在 FoxPro 和 .NET 之间传递一个数组

```
loPersons = loBridge.Invokemethod(loFox,"GetPersons") && ComArray

? StringFormat("Initial Array Size: {0}", loPersons.Count)

*** Create a new Person
loNewPerson = loPersons.CreateItem()

loNewPerson.FirstName = "Billy"
loNewPerson.LastName = "Nobody"
loNewPerson.Entered = Datetime()
loNewPerson.Address.Street = "121 Nowhere lane"

*** Add the person to the array
loPersons.AddItem(loNewPerson)

? StringFormat("After add array Size: {0}", loPersons.Count)

*** Pass the array back to .NET
loBridge.Invokemethod(loFox,"AcceptPersons",loPersons)

? StringFormat(".NET Persons Array Size after update: {0}",
loBridge.GetPropertyEx(loFox,"Persons.Count"))
```

代码首先检索Persons列表并将结果作为ComArray实例检索。 如果您记得ComArray是一个.NET数组的COM包装器，它具有一个包含.NET数组的实例属性和一组可以操纵数组的方法。

我们注意到从我们之前的演示开始的2人阵列的计数。 然后我调用loPersons.CreateItem（）来创建一个新的Person对象。 CreateItem创建一个新的数组元素类型的.NET实例 - 在这种情况下是一个Person对象。 该对象使用值填充，并使用AddItem（）方法添加到loPersons ComArray中。

最后，我们调用AcceptPersons并将ComArray实例传递给.NET方法。 wwDotnetBridge自动修复ComArray并在内部将实例变量传递给服务器。 这

种魔法只适用于间接方法（在这种情况下为InvokeMethod） - 它不适用于直接访问。 如果您尝试调用:

```
? loFox.AcceptPersons(loPersons)
```

你会得到一个错误，因为目标方法不接受一个ComArray对象。 你也不能这样做:

```
loFox.AcceptPersons(loPersons.Instance)
```

这是行不通的，因为一旦你引用实例它就会被转换成一个FoxPro数组，并且不能再传递给.NET。 InvokeMethod() 是完成这项工作所必需的。

可枚举的.NET 类型和 ComArray

.NET有一个用于展示可枚举集合的IEnumerable接口。 IEnumerable用于ForEach迭代，但它也是一种以增量方式表示集合数据的机制。 不是将数据加载到数据结构中，而是一次读取数据并提供一个项目，这可以是非常有效的。

但是，IEnumerable没有相应的COM接口，因此一次加载一个的枚举类型会因资源错误而失败。

ComArray类有一个漂亮的帮助器，但可以通过FromEnumerable（）方法在某些场景中提供帮助。 这方面的一个例子是被定义为泛型类型的示例类中的Persons成员

```
public List<Person> Persons { get; set; }
```

这样是无法直接从FoxPro访问的。但是，List <t> 实现 IEnumerable 并且作为 IEnumerable的一部分，您可以使用 ToArray() 将一个枚举转换为数组。通过这样做，您可以在 FoxPro 中收到结果。

```
loPersonArray= loBridge.CreateArray()
```

```
loPersonArray.FromEnumerable(loFox.Persons)
```

现在你可以开启存储在 loPersonArray 中的 ComArray 数组了。

DataSet 转换

wwDotnetBridge 还包含一些 DataSet 转换例程，可以非常容易地接受并传回 DataSet 和游标结果。

如果您有一个返回 .NET DataSet 的方法:

```
public DataSet GetWebLogEntries()
{
    var sql = new SqlDataAccess("server=.;database=Weblog;integrated security=true");

    return sql.ExecuteDataSet("TWebLog",
        "select top 20 * from blog_entries where entrytype=@entryType order by entered Desc",
        sql.CreateParameter("@entryType",1));
}
```

此代码使用 .NET 工具包中的 Westwind.Utilities.dll 中的 SqlDataAccess 类对 SQL Server 执行查询，并将结果作为 DataSet 返回。

您现在可以在 Foxpro 中接收该数据集并将其轻松转换为一个或多个游标（如果该 DataSet 包含多个表）：

```
loDS = loFox.GetWebLogEntries()
loBridge.DataSetToCursors(loDS)
SELE TWEBLOG
BROWSE
```

如果您愿意，也可以使用 XmlAdapter 而不使用游标，以便使用 DataSetToXmlAdapter() 方法操纵数据或控制将哪些表转换为游标。

要发送一个 FoxPro 游标到 .NET，您可以使用以下命令:

```
Create Cursor TPersons (FirstName c(30), LastName c(30),Entered T)

Insert Into TPersons (FirstName,LastName,Entered) Values;
                    ("Rick","Strahl",Datetime())
```

```
Insert Into TPersons (FirstName,LastName,Entered) Values;
                    ("Markus","Egger",Datetime())

Browse

loDs = loBridge.CursorToDataSet("TPersons")
? loFox.AcceptDataSet(loDs)
```

West Wind Technologies 的 wwDotnetBridge

我几年来一直在几个应用程序中使用 wwDotnetBridge，这对于为我扩展 FoxPro 应用程序的生命周期一直是一种救命手段。 它允许我以一种难以或根本不可能的方式与 .NET 集成多个应用程序。

West Wind Html Help Builder

Html Help Builder 是一个帮助和文档生成工具，可以轻松地为开发人员和最终用户或其他技术文档生成帮助文件，联机文档和Word输出。

Help Builder 支持导入 .NET 类和程序集，并根据 .NET DLL 文件中包含的元数据自动记录它们。它允许我从 .NET 组件提供相当完整的文档。内部帮助生成器使用 .NET 反射检索类型导入信息。

为了使这项工作成为可能，我构建了一个自定义的 .NET 组件，该组件可处理所有类型的导入信息，创建一个 FoxPro 友好的 .NET 结构，然后将其传递回 FoxPro 以便轻松解析并输出到帮助文档中。这个封装中有很多逻辑 - 它解析类型，处理来自 XML 文档文件的合并数据以及遍历类型层次结构，修复用于自动链接的类型引用等等。这是一段复杂的代码，但 FoxPro 接口非常简单，仅以带有属性和数组的类的形式查看最终结果，并且没有方法。

Help Builder 还处理 Visual Studio RTF 文本通过一些我前面挖出的 .NET 代码导入并集成到帮助生成器 NET 帮助程序集中。有一些相当棘手的 .NET 代码修复了 .NET 中的 HTML 文本，最后 FoxPro 接口是对类的单个方法调用 - 所有的逻辑都是在.NET内部抽象出来的。

Help Builder 还可以导入现有的 CHM 文件，并使用名为 Html Agility Pack 的 .NET 库来执行此操作。Agility pack 有能力读取 CHM 帮助文件存储格式并检索单个帮助文件，因为它是一个轻量级 HTML DOM 解析器允许我解析生成的 HTML 文档，包括选取图像和资源链接并从旧的帮助文件。我再次使用一个小的 .NET 封装来执行 Html Agility Pack 的自动化，然后从 FoxPro 调用该代码。

所有这三种情况对 Help Builder 中的功能都非常重要，.NET 集成使其成为可能。在所有这三种情况下，我使用包装类将我需要的 .NET 功能抽象为在 FoxPro 中使用更容易的东西。

West Wind Web Service Proxy Generator

The Web Service Proxy Generator 是一个使用 .NET 作为中介动态创建 SOAP Web 服务的 FoxPro 代理的工具。它使用向导界面导入 Web 服务并创建一个 .NET 代理和访问 .NET 代理的 FoxPro 类。wwDotnetBridge 是该产品的重要组成部分，因为它用于在没有 COM 注册的情况下从 FoxPro 访问生成的 .NET 代理。它在向导中用于处理由 .NET 工具生成的 .NET 代理类的编译，更重要的是由生成的代理类进行编译。 FoxPro 类对 .NET Web Service 代理类进行直接调用。

wwDotnetBridge 对于允许访问从 Web 服务返回的无数类型（尤其是 Web 服务中非常常见的数组）非常重要。我见过的一些最复杂的 wwDotnetBridge .NET 访问场景涉及深层 Web 服务结果层次结构。毫不奇怪，这个工具激发了 wwDotnetBridge 的许多增强功能，因为不同的用户问题使我能够捕获许多常见用例并将其抽象化。

总结

COM 互操作可能是 Visual FoxPro 中使用最多的扩展功能之一。 .NET 提供了大量的功能，可用于 .NET 框架本身的许多内置功能，第三方库和各种可用的开源库。 wwDotnetBridge打开了这个 .NET 市场的大部分，只需很小的复杂程度即可从 FoxPro 访问。

COM 互操作有许多限制，但 wwDotnetBridge 提供了许多帮助功能，可让您直接访问 .NET的大部分功能。

wwDotnetBridge 现在免费且开放源代码，代码可在线获取。 它也是 West Wind Web Connection 和 West Wind Internet Client Toolkit 的一部分，它是官方支持的一套工具的一部分。

资源

wwDotnetBridge Home Page
Samples and Source Code

wwDotnetBridge on GitHub
Using .NET COM Components from Visual FoxPro
Open Pop
Red Gate's .NET Reflector