

在.NET 平台下实现跨语言开发

刘 艺

(本文发表在《程序员》杂志 2005 年第 6 期)

日前读到很多软件公司招聘.NET 程序员的广告,要求程序员统一使用某种编程语言进行开发,觉得十分可笑。既然是在.NET 平台下开发,那么使用 C#、VB.NET、Delphi.NET、J#或别的什么语言究竟有会多大的差别?.NET 平台将不同语言的代码统统编译成中间语言(IL),在虚拟机上运行。于是一个程序的好坏已经不是由语言的“高低贵贱”所决定,而是真正由程序员的水平所决定。因此,招聘到什么水平的程序员,比招到使用什么语言的程序员更重要。对于一个熟练的程序员,选择使用什么语言写程序已经完全是个人的偏好,这种偏好应该在支持跨语言开发的.NET 平台下得到尊重和鼓励,以充分发挥程序员的个性和创造力。

实现跨语言的开发,对于微软一直是一个梦想。微软早几年推出的 COM 就希望不同语言能够通过开发 COM 组件来进行实现跨语言的功能调用和代码重用。基于 COM 的跨语言开发听起来不错,但是具体实现起来却比想象的还要难,让人望而却步!

传说中,远古的人类为了上天堂,召集了千万人马修筑通天塔,工程浩大但进展顺利。上帝知道了后大为震惊,决定立即中止人类的计划。于是,他制造了很多不同的语言给人类使用。终于,由于语言不通,人类无法协调工作,建造通天塔的计划不得不半途而废。虽然上帝为了阻止人们建造通天塔而为人类设置了不同语言的,造成了交流的障碍,但这并不妨碍人们尝试各种方法突破语言障碍的努力。在计算机语言层出不穷的年代,微软的.NET 平台为使用不同语言的程序员提供了一个比较好的解决方案。

微软的.NET 是一种软件框架,可用于构建和运行不同类型的软件。.NET 框架的多语言功能使开发人员可以使用最适合给定任务或技能级别的编程语言,并将这些语言合并到一个应用程序中。用不同语言编写的组件可以透明地利用每个组件的功能,而开发人员也无需进行任何其他额外的工作,从而使开发人员将精力都集中到编写核心业务逻辑代码当中。目前已有 20 多种商业性和学术性编程语言宣布支持.NET 平台,其中包括 APL、Visual Basic、C#、C++、COBOL、Eiffel、Forth、Fortran、Java、J#、Prolog、Pascal、Delphi、Perl、Python 和 RPG。

那么在.NET 平台下如何实现跨语言的开发呢?下面我就通过具体的开发实例来进行介绍。

我要演示的跨语言开发实例是一个果园系统。首先我们来对该系统进行面向对象的分析和设计。该果园系统的需求为:

- 1、园丁投资并种植各种水果;
- 2、收获水果,并计算收益。

该系统的用例图如图 1 所示,为了便于说明问题,我选用的这个该果园系统例子并不复杂。

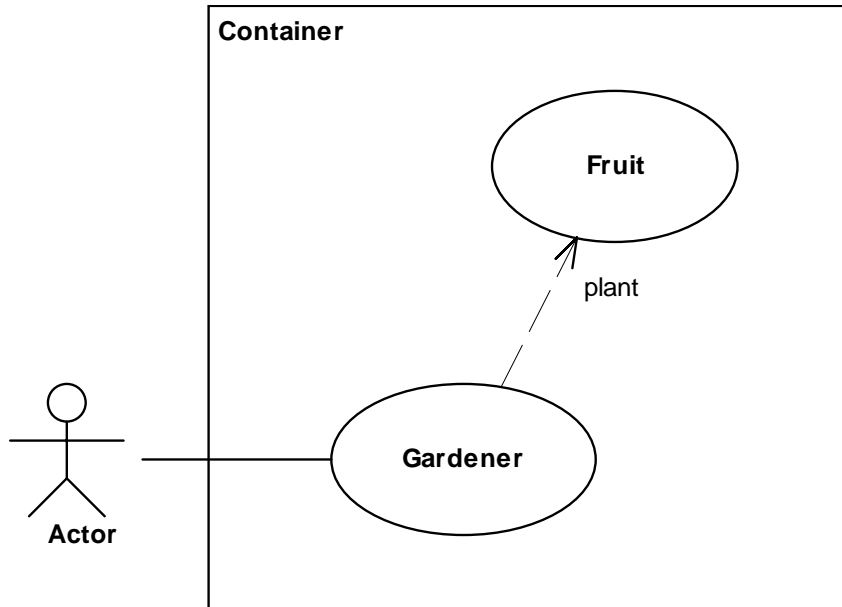


图 1 果园系统的用例图

进一步分析系统，我们可以设计出如图 2 所示的果园系统类图，这里包含了 TGardener、TFruit、TBerry、TTropicalFruit、TCitrusFruit 这几个类，其中 TGardener 与 TFruit 是关联关系，TFruit 和 TBerry、TTropicalFruit、TCitrusFruit 之间是继承关系。

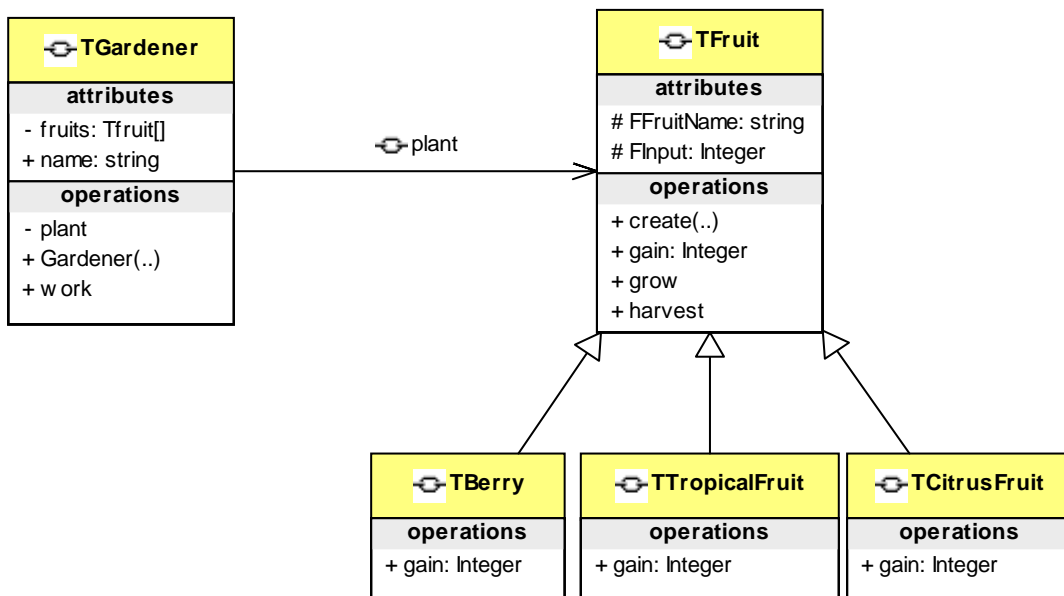


图 2 果园系统的类图

为了演示跨语言的合作开发，我们的实现方案可以设计成如图 3 所示的架构。即某个 Delphi 程序员用 Delphi.NET 开发出 Fruit 配件 (fruitLib.dll)，某个 C#程序员用 C#开发出 Gardener 配件 (GardenerLib.dll)，然后作为公共组件供客户程序调用。所谓配件是.NET 应用程序和库的离散部署模块，其中包含了.NET 的中间语言 (IL) 代码。配件可以是可执行模块 (.exe 文件) 或者是共享的库 (.dll 文件)。这里调用配件的客户程序既可以是另一个配件 (如 Gardener 配件调用 Fruit 配件)，也可以是主程序 (如调用 Gardener 配件的 Program

主程序)。但对于客户程序是用何种语言开发的并没有指定，只要是.NET 兼容的语言既可。本文最后会用 J#程序来调用配件，演示果园系统的运行效果。

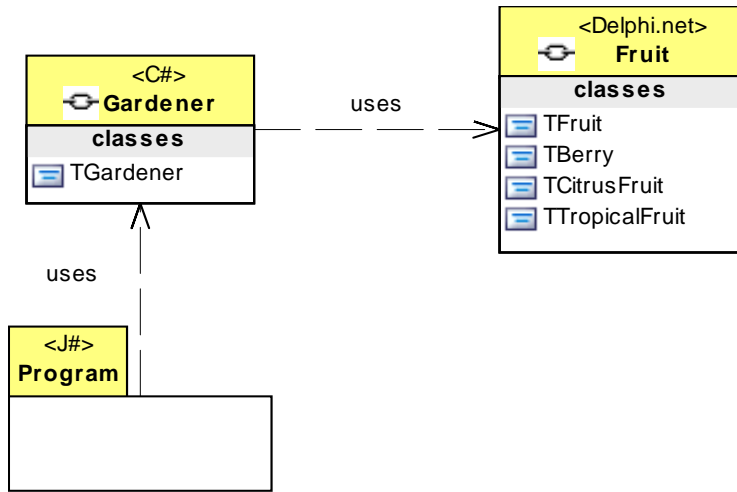


图 3 果园系统的实现图

接下来，我们的 Delphi 程序员先用 Delphi 2005 来创建一个 Fruit 配件 (fruitLib.dll)。在 Delphi 2005 集成开发环境 (IDE) 中点击工具栏上的 New Items 按钮，弹出如图 4 所示的 New Items 对话框。在对话框左边选择 Delphi for .NET Projects 类别，右边选择 Package 项目，系统会自动创建一个 Package 项目，如图 5 左图所示。

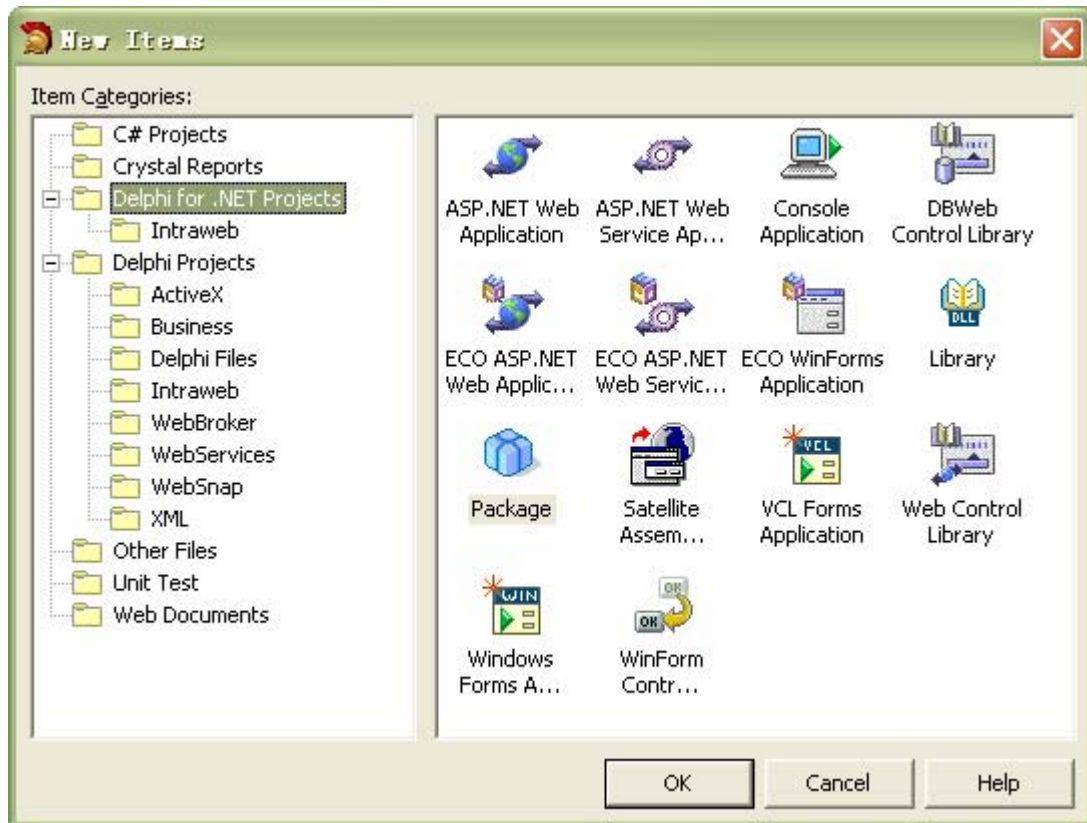


图 4 通过 Delphi2005 的 Package 向导创建 Fruit 配件 (fruitLib.dll)

注意，New Items 对话框中的 Package 和 Library 比较容易搞混淆。这里的 Package 已经不是传统 Delphi 中包的概念了，而是完全匹配.NET 的配件，相当于 C#中的 Class Library。这里的 Library 反而是用于创建与 Win32 兼容的配件，不像 Package 配件那样将其所包含单

元中的所有代码都链接进来。

一旦创建好 Package 项目，就可以向其添加代码单元文件。鼠标右键点击 Package1 项目的 Contains 文件夹，选择弹出菜单的 Add... 菜单项，打开 Add 对话框，如图 5 右图所示添加一个已有的单元。或者也可以直接在 Contains 文件夹中新建一个 Delphi 单元文件。

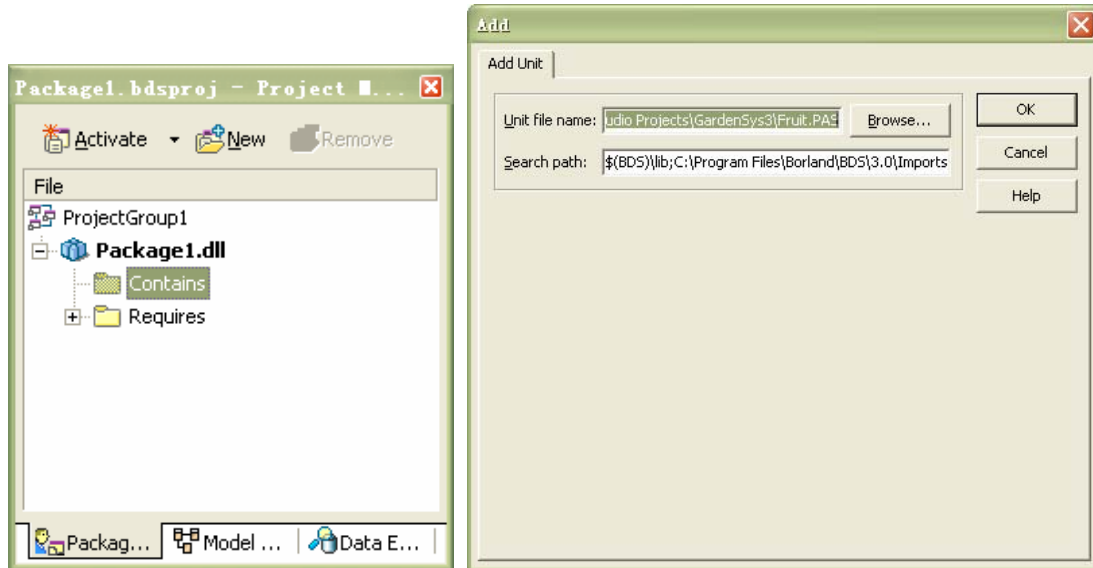


图 5 向 Package 中添加 Delphi 的 Fruit 单元文件

我们将这个在 Package 项目命名为 fruitLib。现在我们在 fruitLib 项目的 Fruit 单元中撰写 TFruit 和 TBerry、TTropicalFruit、TCitrusFruit 几个类，其代码如示例程序 1 所示。

TBerry、TTropicalFruit、TCitrusFruit 都是 TFruit 的派生类，它们继承了 TFruit 类的 create、grow、harvest、gain 方法。但是，因为不同派生类对投资收益有不同的计算方法，所以分别覆盖了基类的 gain 方法，从而在后面的计算中可以实现多态。

调试无误后，我们就可以编译得到 fruitLib.dll 文件，这就是一个标准的.NET 配件。

示例程序 1 Fruit 单元的源代码

```
1: unit Fruit;
2:
3: interface
4:
5: type
6:     TFruit = class (TObject)
7:     protected
8:         FFruitName: string;
9:         FInput: Integer;
10:    public
11:        constructor create(name:string;investment:integer);
12:        function gain: Integer; virtual;
13:        procedure grow;
14:        procedure harvest;
15:    end;
16:
17:    TBerry = class (TFruit)
18:    public
```

```

19:         function gain: Integer; override;
20:     end;
21:
22:     TCitrusFruit = class (TFruit)
23:     public
24:         function gain: Integer; override;
25:     end;
26:
27:     TTropicalFruit = class (TFruit)
28:     public
29:         function gain: Integer; override;
30:     end;
31:
32: implementation
33:
34: { ----- TFruit ----- }
35: constructor TFruit.create(name:string;investment:integer);
36: begin
37:     inherited create;
38:     FFruitName:=name;
39:     FInput:= investment;
40: end;
41:
42: function TFruit.gain: Integer;
43: begin
44:     result:=FInput*2;
45: end;
46:
47: procedure TFruit.grow;
48: begin
49:     writeln(FFruitName+'生长...');
50: end;
51:
52: procedure TFruit.harvest;
53: begin
54:     writeln(FFruitName+'收获...');
55: end;
56:
57: { ----- TBerry ----- }
58: function TBerry.gain: Integer;
59: var
60:     g: Integer;
61: begin
62:     if FInput>20 then

```

```

63:         g:=inherited gain -200
64:     else
65:         g:= 0;
66:     writeln(FFruitName+'投入'+FInput.ToString+'产出'+g.ToString);
67:     result:=g;
68: end;
69:
70: {----- TCitrusFruit -----}
71: function TCitrusFruit.gain: Integer;
72: var
73:     g: Integer;
74: begin
75:     g:=FInput*3 ;
76:     writeln(FFruitName+'投入'+FInput.ToString+'产出'+g.ToString);
77:     result:=g;
78: end;
79:
80: { ----- TTropicalFruit ----- }
81: function TTropicalFruit.gain: Integer;
82: var
83:     g: Integer;
84: begin
85:     g:=inherited gain ;
86:     writeln(FFruitName+'投入'+FInput.ToString+'产出'+g.ToString);
87:     result:=g;
88: end;
89:
90: end.

```

现在轮到 C#程序员用 C#开发 Gardener 配件 (GardenerLib.dll) 的时候了。这个 Gardener 配件显然包括了 TGardener 类, 不过 C#程序员更喜欢用 C#的习惯来命名一个不用 T 作前缀的类 Gardener。当然 Gardener 类需要用到 (关联) TFruit 类及其派生类, 因此需要引用 Fruit 配件 (fruitLib.dll)。

创建 C#类库 (Class Library) 的工具很多, 可以用微软的 Visual C#.NET 或 Borland 的 C# Builder。通过这两个开发工具都可以方便地添加 fruitLib.dll 引用, 他们的界面分别如图 6 和图 7 所示, 操作也很相似。

如果我们在 C#类库 GardenerLib 项目中察看 fruitLib.dll 引用中的类信息, 可以看到其中可用的一些类和类成员, 如图 8 所示 (以微软的 Visual C#.NET 为例)。



图 6 微软 Visual C#.NET 添加 fruitLib.dll 引用的界面

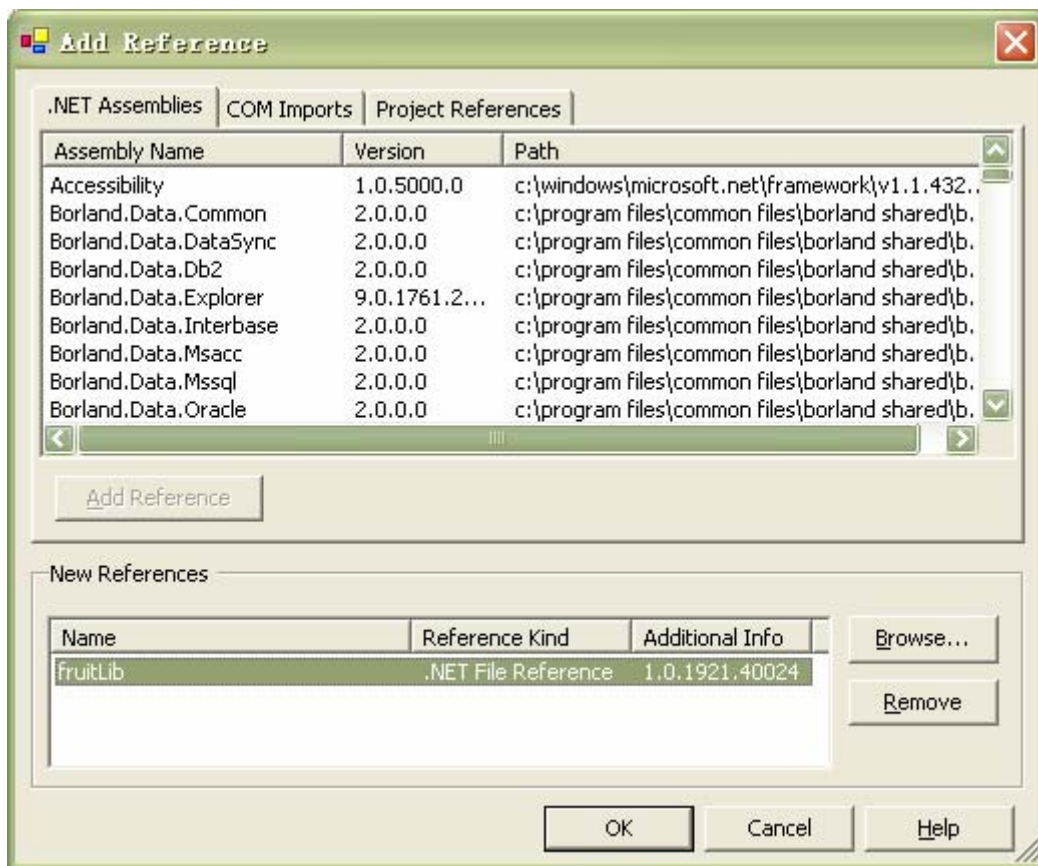


图 7 Borland 的 C# Builder 添加 fruitLib.dll 引用的界面

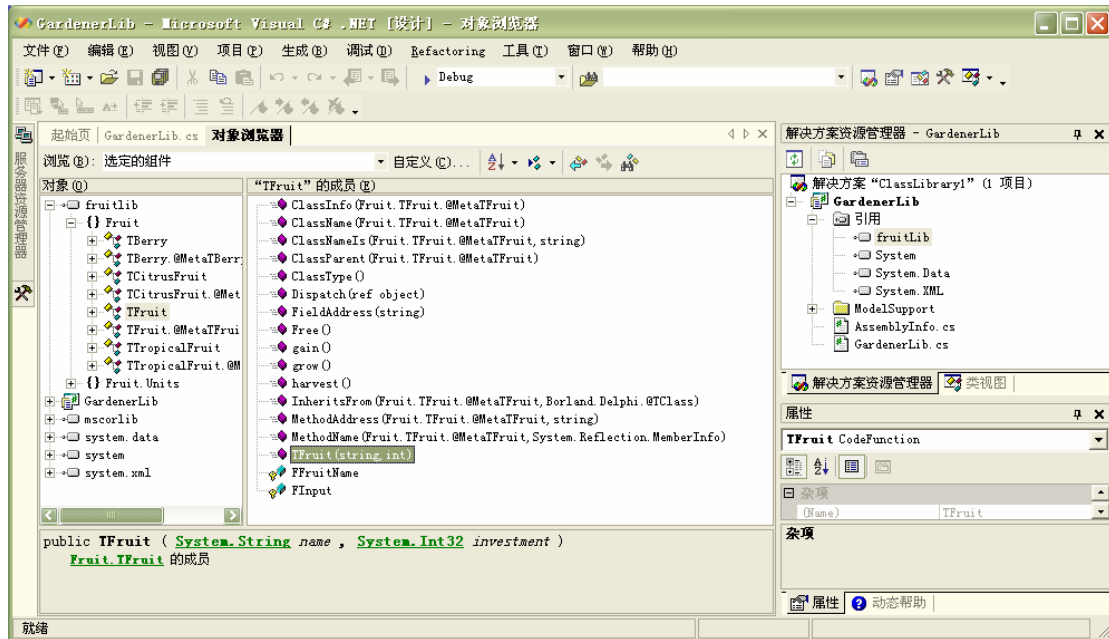


图 8 在 C# 类库 GardenerLib 项目中察看 fruitLib.dll 引用中的类信息

示例程序 2 给出了 C# 类库 GardenerLib 中的 Gardener.cs 源代码。注意在代码第 2 行，using 了 Fruit，这样我们就可以在程序中直接使用这个 Fruit 配件（fruitLib.dll）中的类了。但是，由于在创建类实例时不同语言上的差异，我们不能使用 Delphi 的构造函数（Create），而是要改用 C# 的 new 来创建类的实例，如代码的第 23-27 行所示。

示例程序 2 C# 类库 GardenerLib 中的 Gardener.cs 源代码

```

1: using System;
2: using Fruit; //来自引用的 fruitLib.dll
3:
4: namespace GardenerLB
5: {
6:     /// <summary>
7:     /// 园丁类
8:     /// </summary>
9:     public class Gardener
10:    {
11:        public string name;
12:        private TFruit[] fruits;
13:
14:        public Gardener(string n)
15:        {
16:            name="勤劳的园丁"+n;
17:        }
18:
19:        private void plant()
20:        {
21:            Console.WriteLine("-----");
22:            fruits=new TFruit[5] ;

```

```

23:         fruits[0]=new TTropicalFruit("香蕉",1000);
24:         fruits[1]=new TBerry("葡萄",2000);
25:         fruits[2]=new TTropicalFruit("菠萝",2000);
26:         fruits[3]=new TBerry("草莓",1000);
27:         fruits[4]=new TCitrusFruit("橘子",1000);
28:         for (int i=0;i!=fruits.Length ;i++)
29:         {
30:             fruits[i].grow();
31:             fruits[i].harvest();
32:         }
33:         Console.WriteLine("-----");
34:     }
35:
36:     public void work()
37:     {
38:         int sum=0;
39:         //种植果树
40:         plant();
41:         //统计收益
42:         for(int i=0;i!=fruits.Length ;i++)
43:         {
44:             sum+=fruits[i].gain();
45:         }
46:         Console.WriteLine("-----");
47:         Console.WriteLine("果园总产出:"+sum.ToString());
48:     }
49: }
50: }

```

这个 C# 类库项目编译后得到的 `GardenerLib.dll` 文件就是我们需要的 `Gardener` 配件。

最后，我们用 `J#`编写一个控制台程序来调用前面的配件，测试一下跨语言开发的实际效果。

首先加入 `Gardener` 配件 (`GardenerLib.dll`) 的引用，然后我们撰写一个带有 `main` 方法的 `Program` 类。该程序如示例程序 3 所示。这段程序除了不能编译成 `Java` 语言字节码格式 (即 `.class` 文件) 外，它和标准的 `Java` 程序几乎没有什么两样。注意在代码第 4 行，`import` 了 `GardenerLB`，这样我们就可以在程序中直接使用 `Gardener` 配件 (`GardenerLib.dll`) 中的类了。

示例程序 3 `J#`版的果园系统源程序

```

1: package GardenSys;
2:
3: import java.io.*;
4: import GardenerLB.*;
5:
6: /**

```

```

7:  * Program 是果园系统的主程序。
8:  */
9:  public class Program
10: {
11:     /** @attribute System.STAThread() */
12:     public static void main(String[] args) throws IOException
13:     {
14:         //在此处添加代码以启动应用程序
15:         BufferedReader input=
            new BufferedReader(new InputStreamReader(System.in));
16:         System.out.println("输入园丁的姓名");
17:         String inputName = input.readLine();
18:         Gardener theGardener = new Gardener(inputName);
19:         String note = theGardener.name;
20:         System.out.println(note + "开始工作");
21:         theGardener.work();
22:         System.out.println("按 Enter 键退出");
23:         inputName = input.readLine();
24:     }
25: }

```

该程序运行结果如图 9 所示，和我们预想的一样。

图 9 J#版的果园系统的运行结果

通过以上的实例可以看出，在.NET 平台下实现跨语言的开发比我们想象的还要容易。为什么.NET 平台下可以如此轻松实现跨语言的开发，这要从 CLS（Common Language Specification，公共语言规范）的高度来理解。

鉴于.NET平台通过CLS为不同的开发语言提供了一个编写配件的标准,只要某种语言(如:Delphi.NET)编写的配件(DLL)符合CLS规则,那么编译后的配件就能够被任何其他支持.NET的语言使用。支持CLS的核心是CTS(Common Type System,通用类型系统),它提供了所有与CLS兼容的语言的一组类型。因此,程序员能够使用他们偏爱的任何与CTS/CLS兼容的语言,并且可以用这些语言创建与CTS/CLS兼容的类型和库,使得不同开发语言所贡献的代码能够实现更大范围的重用与合作。

公共语言运行库通过指定和强制公共类型系统以及提供元数据为语言互用性提供了必要的基础。因为所有面向运行库的语言都遵循通用类型系统规则来定义和使用类型,类型的用法在各种语言之间是一致的。元数据通过定义统一的存储和检索类型信息的机制使语言互用性成为可能。编译器将类型信息存储为元数据,公共语言运行库使用该信息在执行过程中提供服务;因为所有类型信息都以相同的方式存储和检索,而与编写该代码的语言无关,所以运行库可以管理多语言应用程序的执行。

在本文所示的果园系统开发实例中,Program类、Gardener类、还有TFruit类及其派生类,分别由J#、C#和Delphi等3种不同的语言撰写。要实现这些对象间的完全交互,而不管这些对象是以何种语言实现的,这些对象必须只向调用方公开那些它们必须与之交互的所有语言的通用功能。为此他们需要遵循公共语言规范(CLS),该规范是跨语言应用程序所需的一套基本语言功能,它确保了语言的互用性。

为了编写符合CLS的代码,我们要注意在以下几处以符合CLS的方式公开功能:

- 公共类的定义。
- 公共类中公共成员的定义,以及派生类可以访问的成员的定義。
- 公共类中公共方法的参数和返回类型,以及派生类可以访问的方法的参数和返回类型。

当然,我们在私有类的定义中、在公共类上私有方法的定义中以及在局部变量中使用的功能完全不必遵循CLS规则。这样我们就可以在实现自己的类的代码中使用任何想要的语言功能并让它仍是一个符合CLS的配件。

由此可见,要充分发挥跨语言开发的优势,关键还在于设计符合CLS的公共接口,从而提高不同语言代码的互用性,最大范围地实现代码重用。

作者简介:

刘艺 海军工程大学副教授,计算机技术作家,Borland产品专家。已出版《Delphi模式编程》、《C# Builder 新锐体验》等10多部计算机专著。个人网站: <http://www.liu-yi.net>