Lua与C#交互

一：LuaInterface介绍

LuaInterface 是 Lua 语言和 Microsoft.NET 平台公共语言运行时 (CLR) 之间的集成库。很多语言已经有面向 CLR 编译器和 CLR 实现，已经存在为微软windows、 BSD 操作系统和 Linux 操作系统。主要用于游戏的热更新操作。

二：测试环境

在VS2012中建一个C#控制台应用程序，并添加LuaInterface.dll的引用。具体操作如课上所示。

VS是一款Window常用的编辑器，可以使用其开发众多语言的项目，企业应用广泛，大家尽量熟悉他的常见操作。

三：LuaInterface具体使用

3.1 常见操作

LuaInterface.Lua类是CLR访问Lua解释器的主要接口，一个LuaInterface.Lua类对象就代表了一个Lua解释器（或Lua执行环境），Lua解释器可以同时存在多个，并且它们之间是完全相互独立的。

下面的简单代码展示了以下功能：

（1）CLR访问Lua的全局域：下标/索引操作[]

（2）CLR新建Lua的table：NewTable

（3）CLR中执行Lua脚本代码或脚本文件：DoFile、DoString

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using LuaInterface;
```

```
namespace TestCSharpAndLuaInterface
{
        static void Main(string[] args)
        {
            // 新建一个Lua解释器，每一个Lua实例都相互独立
            Lua lua = new Lua();

            // Lua的索引操作[]可以创建、访问、修改global域，括号
里面是变量名
            // 创建global域num和str
            lua["num"] = 2;
            lua["str"] = "a string";

            // 创建空table
            lua.NewTable("tab");

            // 执行lua脚本，着两个方法都会返回object[]记录脚本的
执行结果
            lua.DoString("num = 100; print(\"i am a lua
string\")");
            lua.DoFile("C:\\luatest\
\testLuaInterface.lua");
            object[] retVals = lua.DoString("return
num,str");

            // 访问global域num和str
            double num = (double)lua["num"];
            string str = (string)lua["str"];

            Console.WriteLine("num = {0}", num);
            Console.WriteLine("str = {0}", str);
            Console.WriteLine("width = {0}",
lua["width"]);
            Console.WriteLine("height = {0}",
lua["height"]);
        }
    }
}
```

LuaIntrface自动对应Lua和CLR中的一些基础类型：

[nil, null]

[string, System.String]

[number, System.Double]

[boolean, System.Boolean]

[table, LuaInterface.LuaTable]

[function, LuaInterface.LuaFunction]

以上对应关系反之亦然。

3.2 Lua调用C#函数

RegisterFunction方法用来将CLR函数注册进Lua解释器，供Lua代码调用。

```
namespace TestCSharpAndLuaInterface
{
    class TestClass
    {
        private int value = 0;

        public void TestPrint(int num)
        {
            Console.WriteLine("TestClass.TestPrint Called!
value = {0}", value = num);
        }

        public static void TestStaticPrint()
        {
            Console.WriteLine("TestClass.TestStaticPrint
Called!");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Lua lua = new Lua();

            TestClass obj = new TestClass();
            // 注册CLR对象方法到Lua，供Lua调用
            lua.RegisterFunction("LuaTestPrint", obj,
obj.GetType().GetMethod("TestPrint"));    // 也可用
typeof(TestClass).GetMethod("TestPrint")
            // 注册CLR静态方法到Lua，供Lua调用
            lua.RegisterFunction("LuaStaticPrint", null,
typeof(TestClass).GetMethod("TestStaticPrint"));

            lua.DoString("LuaTestPrint(10)");
            lua.DoString("LuaStaticPrint()");
        }
    }
}
```

3.3 CLR调用Lua

在外部单独编写lua代码文件，然后在C#工程中使用lua.DoFile接口运行lua代码。这种方式比较灵活并且能够更方便的测试LuaInterface所提供的各项功能，我们后面的测试代码均是在这种模式系下进行测试。

这种模式下就不需要在lua脚本中手动require "luanet"了，因为已经手动将LuaInterface的引用添加到工程中了，lua脚本中直接使用luanet就可以访问各接口了。

luanet.load_assembly函数：加载CLR程序集；

luanet.import_type函数：加载程序集中的类型；

luanet.get_constructor_bysig函数：显示获取某个特定的构造函数；

C#主要代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using LuaInterface;

namespace TesLuaInterface
{
    class TestClass2
    {
        public TestClass2(string str)
        {
            Console.WriteLine("called TestClass2(string
str) str = {0}", str);
        }

        public TestClass2(int n)
        {
            Console.WriteLine("called TestClass2(int n) n
= {0}", n);
        }
```

```
            public TestClass2(int n1, int n2)
            {
                Console.WriteLine("called TestClass2(int n1,
    int n2) n1 = {0}, n2 = {1}", n1, n2);
            }
        }

            // 加载和实例化CLR类型
            static void Main(string[] args)
            {
                Lua lua = new Lua();

                lua.DoFile("C:\\luatest\\testLuaNet.lua");
            }
        }
    }
```

Lua代码如下所示：

```
-- 加载自定义类型，先加载程序集，在加载类型
luanet.load_assembly("TestEnvi")
TestClass = luanet.import_type("TesLuaInterface.TestClass2")

obj1 = TestClass(2, 3)    -- 匹配public TestClass2(int n1, int
n2)
obj2 = TestClass("x")    -- 匹配public TestClass2(string str)
obj3 = TestClass(3)        -- 匹public TestClass2(string str)

TestClass_cons2 = luanet.get_constructor_bysig(TestClass,
'System.Int32')
obj3 = TestClass_cons2(3)    -- 匹配public TestClass2(int n)
```

TestEnvi为我建的工程代码的程序集名字，这一项是可以在工程属性中进行设置的，TestLuaInterface为测试代码的命名空间。

从上面的构造函数的匹配可以看出，LuaInterface匹配构造函数的规律：

LuaInterface匹配第一个能够匹配的构造函数，在这个过程中，numerical string（数字字符串）会自动匹配number，而number可以自动匹配string，所以TestClass(3)匹配到了参数为string的构造函数。