xLua 教程

Lua 文件加载

一、执行字符串

最基本是直接用 LuaEnv.DoString 执行一个字符串，当然，字符串得符合 Lua 语法

比如：`luaenv.DoString("print('hello world')")`

完整代码见 XLua\Tutorial\LoadLuaScript\ByString 目录

但这种方式并不建议，更建议下面介绍这种方法。

二、加载 Lua 文件

用 lua 的 require 函数即可

比如：`DoString("require 'byfile'")`

完整代码见 XLua\Tutorial\LoadLuaScript\ByFile 目录

require 实际上是调一个个的 loader 去加载，有一个成功就不再往下尝试，全失败则报文件找不到。

目前 xLua 除了原生的 loader 外，还添加了从 Resource 加载的 loader，需要注意的是因为 Resource 只支持有限的后缀，放 Resources 下的 lua 文件得加上 txt 后缀（见附带的例子）。

建议的加载 Lua 脚本方式是：整个程序就一个 `DoString("require 'main'")`，然后在 main.lua 加载其它脚本（类似 lua 脚本的命令行执行：lua main.lua）。

有童鞋会问：要是我的 Lua 文件是下载回来的，或者某个自定义的文件格式里头解压出来，或者需要解密等等，怎么办？问得好，xLua 的自定义 Loader 可以满足这些需求。

三、自定义 Loader

在 xLua 加自定义 loader 是很简单的，只涉及到一个接口：

```
public delegate byte[] CustomLoader(ref string filepath);

public void LuaEnv.AddLoader(CustomLoader loader)
```

通过 AddLoader 可以注册个回调，该回调参数是字符串，lua 代码里头调用 require 时，参数将会透传给回调，回调中就可以根据这个参数去加载指定文件，如果需要支持调试，需要把 filepath 修改为真实路径传出。该回调返回值是一个 byte 数组，如果为空表示该 loader 找不到，否则则为 lua 文件的内容。

有了这个就简单了，用 IIPS 的 IFS？没问题。写个 loader 调用 IIPS 的接口读文件内容即可。文件已经加密？没问题，自己写 loader 读取文件解密后返回即可。。。

完整示例见 XLua\Tutorial\LoadLuaScript\Loader

C#访问 Lua

这里指的是 C#主动发起对 Lua 数据结构的访问。

本章涉及到的例子都可以在 XLua\Tutorial\CSharpCallLua 下找到。

一、获取一个全局基本数据类型

访问 LuaEnv.Global 就可以了，上面有个模版 Get 方法，可指定返回的类型。

```
luaenv.Global.Get<int>("a")
luaenv.Global.Get<string>("b")
luaenv.Global.Get<bool>("c")
```

二、访问一个全局的 table

也是用上面的 Get 方法，那类型要指定成啥呢？

1、映射到普通 class 或 struct

定义一个 class，有对应于 table 的字段的 public 属性，而且有无参数构造函数即可，比如对于{f1 = 100, f2 = 100}可以定义一个包含 public int f1;public int f2;的 class。

这种方式下 xLua 会帮你 new 一个实例，并把对应的字段赋值过去。

table 的属性可以多于或者少于 class 的属性。可以嵌套其它复杂类型。

要注意的是，这个过程是值拷贝，如果 class 比较复杂代价会比较大。而且修改 class 的字段值不会同步到 table，反过来也不会。

这个功能可以通过把类型加到 `GCOptimize` 生成降低开销，详细可参见配置介绍文档。

那有没有引用方式的映射呢？有，下面这个就是：

2、映射到一个 interface

这种方式依赖于生成代码（如果没生成代码会抛 `InvalidCastException` 异常），代码生成器会生成这个 interface 的实例，如果 get 一个属性，生成代码会 get 对应的 table 字段，如果 set 属性也会设置对应的字段。甚至可以通过 interface 的方法访问 lua 的函数。

3、更轻量级的 by value 方式：映射到 Dictionary<>，List<>

不想定义 class 或者 interface 的话，可以考虑用这个，前提 table 下 key 和 value 的类型

都是一致的。

4、另外一种 by ref 方式：映射到 LuaTable 类

这种方式好处是不需要生成代码，但也有一些问题，比如慢，比方式 2 要慢一个数量级，比如没有类型检查。

三、访问一个全局的 function

仍然是用 Get 方法，不同的是类型映射。

1、映射到 delegate

这种是建议的方式，性能好很多，而且类型安全。缺点是要生成代码（如果没生成代码会抛 `InvalidCastException` 异常）。

delegate 要怎样声明呢？

对于 function 的每个参数就声明一个输入类型的参数。

多返回值要怎么处理？从左往右映射到 c#的输出参数，输出参数包括返回值，out 参数，ref 参数。

参数、返回值类型支持哪些呢？都支持，各种复杂类型，out，ref 修饰的，甚至可以返回另外一个 delegate。

delegate 的使用就更简单了，直接像个函数那样用就可以了。

2、映射到 LuaFunction

这种方式的优缺点刚好和第一种相反。

使用也简单，LuaFunction 上有个变参的 Call 函数，可以传任意类型，任意个数的参数，返回值是 object 的数组，对应于 lua 的多返回值。

四、使用建议

1、访问 lua 全局数据，特别是 table 以及 function，代价比较大，建议尽量少做，比如在初始化时把要调用的 lua function 获取一次（映射到 delegate）后，保存下来，后续直接调用该 delegate 即可。table 也类似。

2、如果 lua 测的实现的部分都以 delegate 和 interface 的方式提供，使用方可以完全和 xLua 解耦：由一个专门的模块负责 xlua 的初始化以及 delegate、interface 的映射，然后把这些 delegate 和 interface 设置到要用到它们的地方。

Lua 调用 C#

本章节涉及到的实例均在 XLua\Tutorial\LuaCallCSharp 下

new C#对象

你在 C#这样 new 一个对象：

```
var newGameObj = new UnityEngine.GameObject();
```

对应到 Lua 是这样：

```
local newGameObj = CS.UnityEngine.GameObject()
```

基本类似，除了：

1、lua 里头没有 new 关键字；

2、所有 C#相关的都放到 CS 下，包括构造函数，静态成员属性、方法；

如果有多个构造函数呢？放心，xlua 支持重载，比如你要调用 GameObject 的带一个 string 参数的构造函数，这么写：

```
local newGameObj2 = CS.UnityEngine.GameObject('helloworld')
```

访问 C#静态属性，方法

读静态属性

```
CS.UnityEngine.Time.deltaTime
```

写静态属性

```
CS.UnityEngine.Time.timeScale = 0.5
```

调用静态方法

```
CS.UnityEngine.GameObject.Find('helloworld')
```

小技巧：如果需要经常访问的类，可以先用局部变量引用后访问，除了减少敲代码的时间，还能提高性能：

```
local GameObject = CS.UnityEngine.GameObject
GameObject.Find('helloworld')
```

访问 C#成员属性，方法

读成员属性

```
testobj.DMF
```

写成员属性

```
testobj.DMF = 1024
```

调用成员方法

注意：调用成员方法，第一个参数需要传该对象，建议用冒号语法糖，如下

```
testobj:DMFunc()
```

父类属性，方法

xlua 支持（通过派生类）访问基类的静态属性，静态方法，（通过派生类实例）访问基类的成员属性，成员方法

参数的输入输出属性（out，ref）

Lua 调用测的参数处理规则：C#的普通参数算一个输入形参，ref 修饰的算一个输入形参，out 不算，然后从左往右对应 lua 调用测的实参列表；

Lua 调用测的返回值处理规则：C#函数的返回值（如果有的话）算一个返回值，out 算一个返回值，ref 算一个返回值，然后从左往右对应 lua 的多返回值。

重载方法

直接通过不同的参数类型进行重载函数的访问，例如：

```
testobj:TestFunc(100)
```

```
testobj:TestFunc('hello')
```

将分别访问整数参数的 TestFunc 和字符串参数的 TestFunc。

注意：xlua 只一定程度上支持重载函数的调用，因为 lua 的类型远远不如 C#丰富，存在一对多的情况，比如 C#的 int，float，double 都对应于 lua 的 number，上面的例子中 TestFunc 如果有这些重载参数，第一行将无法区分开来，只能调用到其中一个（生成代码中排前面的那个）

操作符

支持的操作符有：+，-，*，/，==，一元-，<，<=， %，[]

参数带默认值的方法

和 C#调用有默认值参数的函数一样，如果所给的实参少于形参，则会用默认值补上。

可变参数方法

对于 C#的如下方法：

```
void VariableParamsFunc(int a, params string[] strs)
```

可以在 lua 里头这样调用：

```
testobj:VariableParamsFunc(5, 'hello', 'john')
```

使用 Extension methods

在 C#里定义了，lua 里就能直接使用。

泛化（模版）方法

不直接支持，可以通过 Extension methods 功能进行封装后调用。

枚举类型

枚举值就像枚举类型下的静态属性一样。

```
testobj:EnumTestFunc(CS.Tutorial.TestEnum.E1)
```

上面的 EnumTestFunc 函数参数是 Tutorial.TestEnum 类型的

另外，如果枚举类加入到生成代码的话，枚举类将支持__CastFrom 方法，可以实现从一个整数或者字符串到枚举值的转换，例如：

```
CS.Tutorial.TestEnum.__CastFrom(1)
```

```
CS.Tutorial.TestEnum.__CastFrom('E1')
```

delegate 使用（调用，+，-）

C#的 delegate 调用：和调用普通 lua 函数一样

+操作符：对应 C#的+操作符，把两个调用串成一个调用链，右操作数可以是同类型的 C# delegate 或者是 lua 函数。

-操作符：和+相反，把一个 delegate 从调用链中移除。

Ps：delegate 属性可以用一个 luafunction 来赋值。

event

比如 testobj 里头有个事件定义是这样：`public event Action TestEvent;`

增加事件回调

```
testobj:TestEvent('+', lua_event_callback)
```

移除事件回调

```
testobj:TestEvent('-', lua_event_callback)
```

64 位整数支持

Lua53 版本 64 位整数（long，ulong）映射到原生的 64 未整数，而 luaji 版本 t，相当于 lua5.1 的标准，本身不支持 64 位，xlua 做了个 64 位支持的扩展库，C#的 long 和 ulong 都将映射到 userdata：

1、支持在 lua 里头进行 64 位的运算，比较，打印

2、支持和 lua number 的运算，比较

3、要注意的是，在 64 扩展库中，实际上只有 int64，ulong 也会先强转成 long 再传递

到 lua，而对 ulong 的一些运算，比较，我们采取和 java 一样的支持方式，提供一组 API，详情请看 API 文档。

C#复杂类型和 table 的自动转换

对于一个有无参构造函数的 C#复杂类型，在 lua 侧可以直接用一个 table 来代替，该 table 对应复杂类型的 public 字段有相应字段即可，支持函数参数传递，属性赋值等，例如：

C#下 B 结构体（class 也支持）定义如下：

```
public struct A
{
    public int a;
}

public struct B
{
    public A b;
    public double c;
}
```

某个类有成员函数如下：

```
void Foo(B b)
```

在 lua 可以这么调用

obj:Foo({b = {a = 100}, c = 200})

获取类型（相当于 C#的 typeof）

比如要获取 UnityEngine.ParticleSystem 类的 Type 信息，可以这样

```
typeof(CS.UnityEngine.ParticleSystem)
```

“强”转

lua 没类型，所以不会有强类型语言的“强转”，但有个有点像的东西：告诉 xlua 要用指定的生成代码去调用一个对象，这在什么情况下能用到呢？有的时候第三方库对外暴露的是一个 interface 或者抽象类，实现类是隐藏的，这样我们无法对实现类进行代码生成。该实现类将会被 xlua 识别为未生成代码而用反射来访问，如果这个调用是很频繁的话还是很影响性能的，这时我们就可以把这个 interface 或者抽象类加到生成代码，然后指定用该生成代码来访问：

```
cast(calc, typeof(CS.Tutorial.Calc))
```

上面就是指定用 CS.Tutorial.Calc 的生成代码来访问 calc 对象。