

Unity ToLua & LuaFramework_UGUI学习笔记



CatherinePlans (/u/f796a54054d1) [+关注](#)

2017.08.05 17:52* 字数 8645 阅读 18163 评论 3 喜欢 34 赞赏 1
(/u/f796a54054d1)

由于网上关于Unity Lua学习的资料较少，本人也是刚入门U3D不久，现在项目准备基于LuaFramework用Lua做热更新开发，最近几天看了ToLua官网的文档说明，研究了一下C#与Lua的交互并做了一些尝试，发现实际入手还是遇到了很多配置文件，方法调用的坑，并根据学习资料做了一些整理，方便初学者参考，好了废话不多说，进入正题，由于比较基础大神绕过，不喜勿喷！

基础部分：

解决的问题

更新频繁，而IOS审核长

IOS无法用DLL反射机制去做代码更新

如果无热更新，客户端每次都需要重新下载一次安装包。用户体验不好

解决发布包的容量问题，一切都可以增量下载

原理

首先要清楚Unity的打包原理，也就是AssetBundle的打包机制，他会把prefab打包成asset格式作为传输的数据。

通过校验文件的MD5值来判断是否需要更新，如果需要更新则下载差异文件。

lua属于解释性文件所以能通过www直接下载到本地，通过C#与lua交互，把逻辑写在lua里，从而实现代码热更新。注意记得加密lua。

实现方案

这里我推荐LuaFramework 这个框架（下简称tolua），以上内容他已经完全封装好，包括上述没提到的CG的一些操作。

我们只需要在tolua里面写属于自己模块部分的逻辑就能简单的实现热更新。

Tolua学习

简介



tolua#是Unity静态绑定lua的一个解决方案，它通过C#提供的反射信息分析代码并生成包装的类。它是一个用来简化在C#中集成lua的插件，可以自动生成用于在lua中访问Unity的绑定代码，并把C#中的常量、变量、函数、属性、类以及枚举暴露给lua。它是从tolua衍变而来。从它的名字可以看出，它是集成了原来的tolua代码通过二次封装写了一个C#与tolua(c)的一个中间层。

(/apps/redir
utm_source
banner-click

基础

要想了解tolua#是如何集成的我们需要对C#的一些特性有一些了解，比如了解C#与原生代码交互的方式等。我们假设读者已经对lua和tolua++有一个比较熟悉，我们略过lua与c或者C++的交互方式，主要介绍一些C#的特性，来帮助我们接下来分析tolua#的集成原理。

C#特性

Attribute

Attribute是一种可由用户自由定义的修饰符（Modifier），可以用来修饰各种需要被修饰的目标。特性Attribute的作用是添加元数据。元数据可以被工具支持，比如：编译器用元数据来辅助编译，调试器用元数据来调试程序。Unity以及tolua#中就会用Attribute来帮助做一些事情。

值类型与引用类型

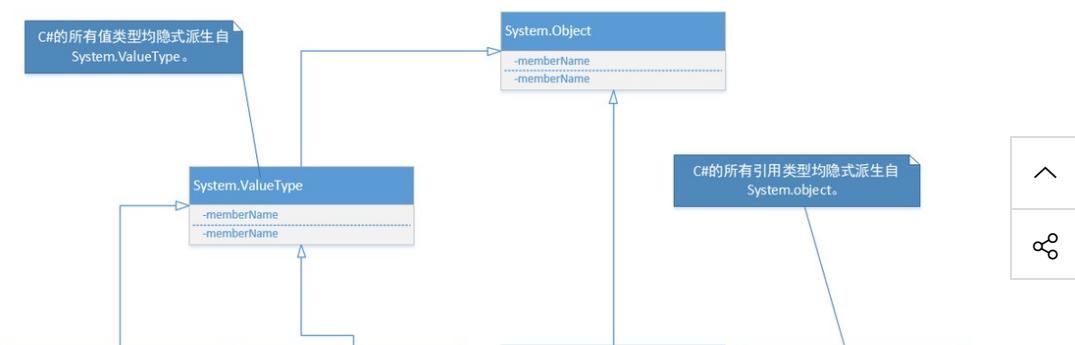
之所以要提这两个概念，是因为很好得理解这两个概念有助于我们写出比较高效的C#代码。

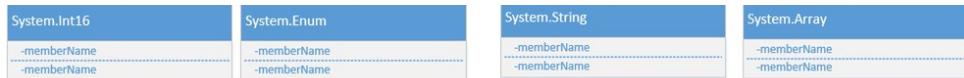
我们知道，C#中的每一种类型要么是值类型，要么是引用类型。所以每个对象要么是值类型的实例，要么是引用类型的实例。

引用类型和值类型都继承自System.Object类。不同的是，几乎所有的引用类型都直接从System.Object继承，而值类型则继承其子类，即直接继承System.ValueType。

作为所有类型的基类，System.Object提供了一组方法，这些方法在所有类型中都能找到，其中包括ToString方法及Clone等方法。

System.ValueType直接继承System.Object，即System.ValueType本身是一个类类型，而不是值类型；System.ValueType没有添加任何成员，但覆盖了所继承的一些方法，使其更适合于值类型。例如，ValueType重写了Equals()方法，从而对值类型按照实例的值来比较，而不是引用地址来比较。





简单了解了值类型与引用类型那么我们下面来看下C#中的装箱和拆箱的概念。

装箱和拆箱

装箱和拆箱是值类型和引用类型之间相互转换是要执行的操作。

1. 装箱在值类型向引用类型转换时发生
2. 拆箱在引用类型向值类型转换时发生

...

```
object objValue = 4;
```

```
int value = (int)objValue;
```

...

上面的两行代码会执行一次装箱操作将整形数字常量4装箱成引用类型object变量objValue；然后又执行一次拆箱操作，将存储到堆上的引用变量objValue存储到局部整形值类型变量value中。

同样我们需要看下IL代码：

...

```
locals init (
```

```
[0] object objValue,
```

```
[1] int32 'value'
```

//上面IL声明两个局部变量object类型的objValue和int32类型的value变量

```
IL_0000: nop
```

```
IL_0001: ldc.i4.4 //将整型数字4压入栈
```

IL_0002: box [mscorlib]System.Int32 //执行IL box指令，在内存堆中申请System.Int32类型需要的堆空间

```
IL_0007: stloc.0 //弹出堆栈上的变量，将它存储到索引为0的局部变量中
```

```
IL_0008: ldloc.0//将索引为0的局部变量（即objValue变量）压入栈
```

IL_0009: unbox.any [mscorlib]System.Int32 //执行IL 拆箱指令unbox.any 将引用类型object转换成System.Int32类型

```
IL_000e: stloc.1 //将栈上的数据存储到索引为1的局部变量即value
```

(/apps/redir
utm_source
banner-clic



...

拆箱操作的执行过程和装箱操作过程正好相反，是将存储在堆上的引用类型值转换为值类型并给值类型变量。

C#调用原生代码

因为tolua#底层库是使用的tolua（c语言编写），那么就需要通过C#来调用原生代码，我们从LuaDll.cs中摘取一段代码来演示如何从C#中调用原生代码

...

```
Public class LuaDll
```

```
{
```

```
[DllImport(LUADLL, CallingConvention = CallingConvention.Cdecl)]
```

```
public static extern void lua_close(IntPtr luaState);
```

```
}
```

...

其中LUADLL对应的字符串就是tolua，在不同的平台上mono会去加载对应的tolua.dll或者tolua.so等文件并调用对应的函数。具体可以参考mono官方的教程。

tolua#集成

tolua#集成主要分为两部分，一部分是运行时需要的代码包括一些手写的和自动生成的绑定代码，另一部分是编辑器相关代码，主要提供代码生成、编译lua文件等操作，具体就是Unity编辑器中提供的功能。用mono打开整个tolua#的工程，

Runtime

Source

Generate 这个文件里面是生成的绑定代码

LuaConst.cs 这个文件是一些lua路径等配置文件。

ToLua

BaseLua 一些基础类型的绑定代码

Core 提供的一些核心功能，包括封装的LuaFunction LuaTable LuaThread LuaState LuaEvent、调用tolua原生代码等等。

Examples 示例

Misc 杂项，目前有LuaClient LuaCoroutine（协程） LuaLooper（用于tick） LuaResolver（用于加载lua文件）

(/apps/redir
utm_source
banner-clic



Reflection 反射相关

Editor

Editor

Custom

CustomSettings.cs 自定义配置文件，用于定义哪些类作为静态类型、哪些类需要导出、哪些附加委托需要导出等。

ToLua

Editor

Extend 扩展一些类的方法。

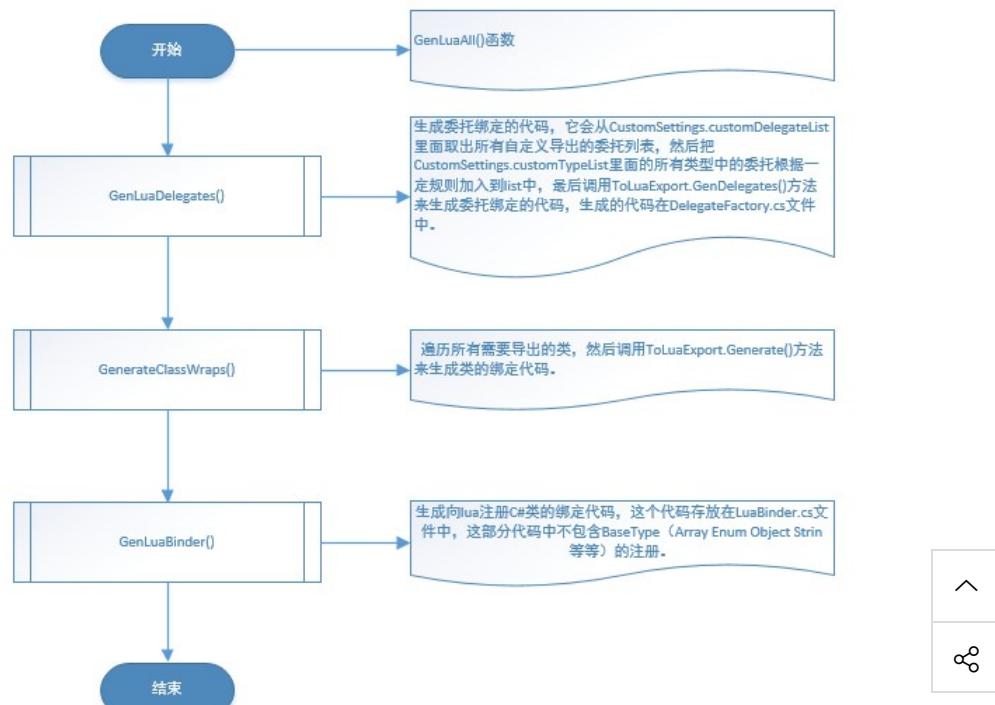
ToLuaExport.cs 真正生成lua绑定的代码

ToLuaMenu.cs Lua菜单上功能对应的代码

ToLuaTree.cs 辅助树结构

Generate All 流程

了解了tolua#的大致文件结构，下面我们来看下tolua#的Generate All 这个功能来分析下它的生成过程。生成绑定代码主要放在ToLuaExport.cs里面，我们并不会对每一个函数进行细致的讲解，如果有什么不了解的地方，可以直接看它的代码。



GenLuaDelegates函数

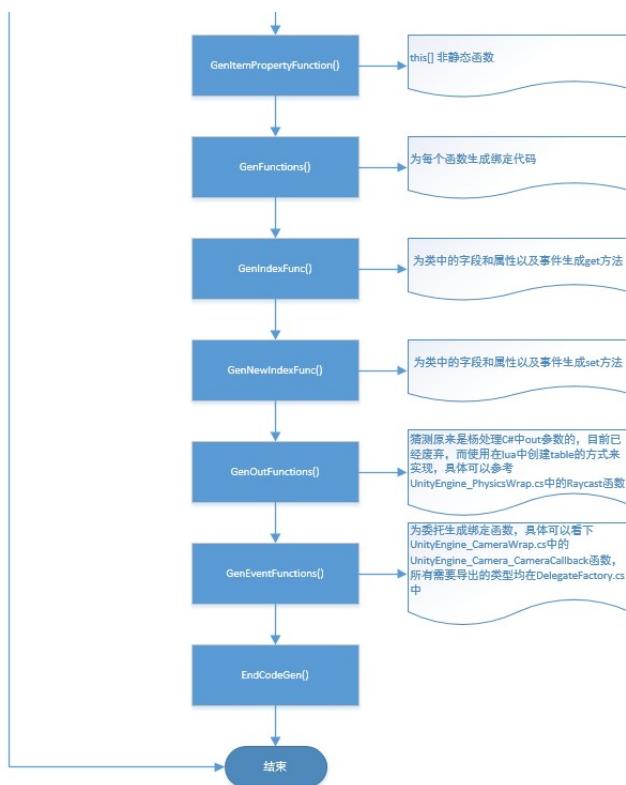
生成委托绑定的代码，它会从CustomSettings.customDelegateList里面取出所有自定义导出的委托列表，然后把CustomSettings.customTypeList里面的所有类型中的委托根据一定规则加入到list中，最后调用ToLuaExport.GenDelegates()方法来生成委托绑定的代码，生成的代码在DelegateFactory.cs文件中。

由于该函数比较简单，我们这里不做展开，可以直接查看ToLuaExport.cs中的GenDelegates()并配合DelegateFactory.cs来查看。

GenerateClassWraps 函数

遍历所有需要导出的类，然后调用ToLuaExport.Generate()方法来生成类的绑定代码。

下面我们来看下ToLuaExport.Generate()方法，其基本流程如下所示：



从上面的流程图我们可以看到，整个过程还是比较清楚的，如果这个类是枚举类型，那么它会调用枚举导出的接口，而如果这个类型是一个普通的类，那么它就会调用上图所示的相应的流程将代码导出。至于结构体类型，目前应该是只支持一些特定的结构体，需要在lua中对应一份实现（Assets\ToLua\Lua目录中），当然它生成的代码也有一些依赖于tolua#的核心运行时，我们前面简单的讲解了如何在编辑器中生成绑定代码，接下来我们讲一下它的核心运行时。

tolua#的核心运行时

tolua# 的运行代码包含SourceàGenerate下面的绑定代码以及ToLuaàBaseType代码以及Core下面的核心代码。接下来我们着重讲一下Core下面的几个主要类。



LuaAttribute.cs

我们前面基础知识部分已经讲过，它在tolua#生成绑定代码时做一些标示使用。

LuaBaseRef.cs

Lua中对象对应C#中对象的一个基类，主要作用是有一个reference指向lua里面的对象，引用计数判断两个对象是否相等等。

ToLua的官方地址 (<https://link.jianshu.com?t=http://www.ulua.org/>)

国内的资料不是太多，所以有什么疑问可以去官网或交流群查询。

本篇文章使用的UGUI + tolua的一个组合，Unity版本必须是5.x，因为Unity5对于之前的版本的ab打包策略更改。

Tolua的UI架构是使用的puremvc，给我的感觉是把简单的东西所复杂化，但是优点在于把所复杂的逻辑解耦成机械操作，其实对于成熟的项目来说，我还是推荐不要舍弃mv c，当然如果实在不喜，也是可以自己修改，或者写插件便于开发。

例子的实现

1.先下载LuaFramework UGUI,我们的项目也是从这个项目所修改而来。

很多东西别人写过的，我就不直接复制过来，而是给出链接，各位自行去看。

2.LuaFramework目录详解 (https://link.jianshu.com?t=http://doc.ulua.org/article/ngui/simpleframework_base1.html)

链接中的目录是ulua的结构目录，其实这是一样的，tolua核心的思想还是和ulua是没差别的。

3.找到Examples里面的,main.scene。

按照LuaFramework的基本用法 (<https://link.jianshu.com?t=http://doc.ulua.org/article/ulua/luaframeworkdejichuyongfa.html>)去进行操作，可以实现热更新。在打包的过程中，我们发现生成了StreamingAssets这个文件夹，这是个可读文件夹。在各个平台都有对应的目录，所以之后所有的更新文件都会在这个目录下，而我们所打包的文件也是通过更新StreamingAssets文件去实现更新迭代的。

lua源码目录介绍：

--3rd：里面是第三方的一些插件lua、实例源码文件，比如：cjson、pbc、pblua、sproto等。

--Common：公用的lua文件目录，如define.lua文件，一些变量声明，全局配置等，functions.lua常用函数库，通讯的protocol.lua协议文件。

--Controller：控制器目录，它不依赖于某一个Lua面板，它是独立存活在Luavm中的一个操作类，操作数据、控制面板显示而已。

--Logic：目录里面存放的是一些管理器类，比如GameManager游戏管理器、NetworkM

(/apps/redir
utm_source
banner-clic



anager网络管理器，如果你有新的管理器可以放到里面。

--System: 这个目录是cstolua的系统目录，里面存放都是一些常用的lua类，为了优化lua调用速度，用lua重写的unity常用类。

--View: 这是面板的视图层，里面都是一些被Unity调用的面板的变量，走的是Unity GameObject的生命周期的事件调用。

(/apps/redir
utm_source
banner-click

lua和C#的交互两个工具类：

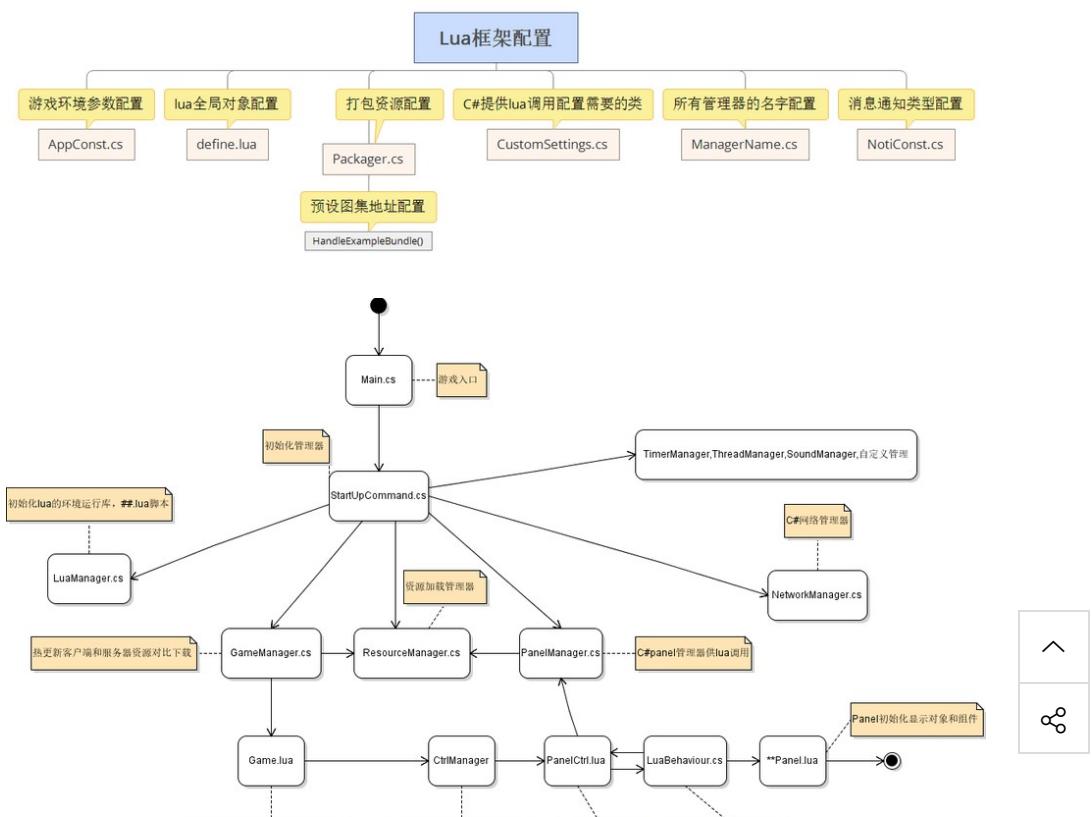
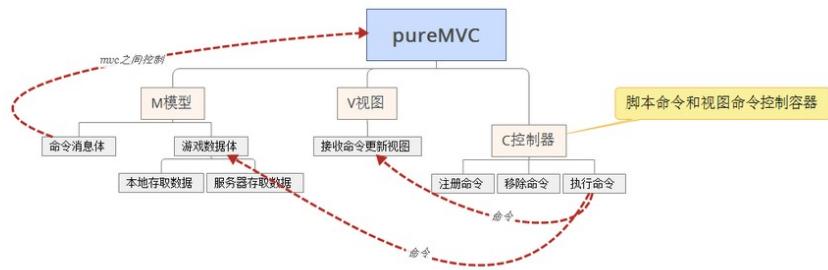
LuaHelper.cs静态类//lua调用C#

Util.cs //静态方法CallMethod C#调用lua

希望大家一起学习一起分享，让lua网络资料更丰富一些！

如果有什么理解的不好不到位，求大神指点！

下面是我做的几个分析图：



lua代码入口，初始化CtrlManager 里面是装着的panel控制脚本集合 Panel的控制处理 *Panel.lua是运行基于它的

应用部分：

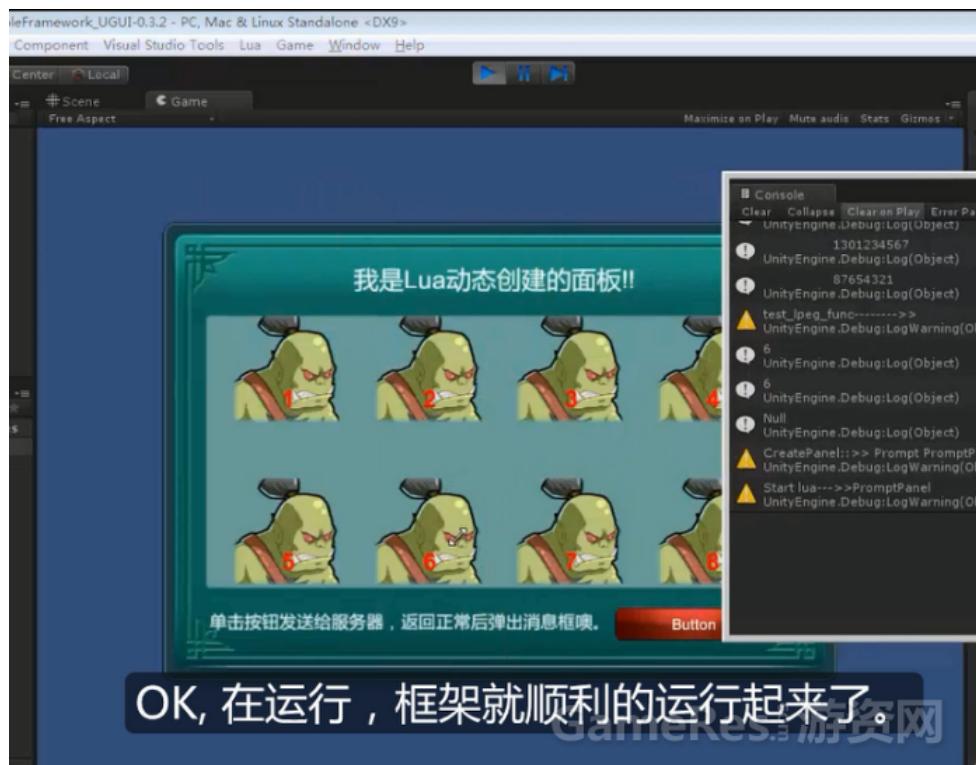
1、安装框架

只要在<http://www.ulua.org/index.html> (<https://link.jianshu.com?t=http://www.ulua.org/index.html>)下载LuaFramework，然后用Unity3D打开，这里用的是LuaFramework_UGUI-1.0.4.109版本以及Unity3D 5.2，其他版本理应相似。打开之后需要点击lua菜单里面的Generate All和LuaFramework菜单里Build XXX Resources，以生成一些必要的文件。

安装过程可以参见<http://pan.baidu.com/s/1gd8fG4N> (<https://link.jianshu.com?t=http://pan.baidu.com/s/1gd8fG4N>)里面的01_uLua_Windows.avi和02_SimpleFramework_UGUI_Windows.avi两个视频（如果在windows系统下）。框架结构请参见http://doc.ulua.org/article/ngui/simpleframework_base1.html (https://link.jianshu.com?t=http://doc.ulua.org/article/ngui/simpleframework_base1.html)，这里不再复述。

若运行后能够弹出示范界面，证明安装成功，可以进入下一步。

(/apps/redir
utm_source
banner-click)



成功运行示范界面（只要客户端能够运行起来就行）

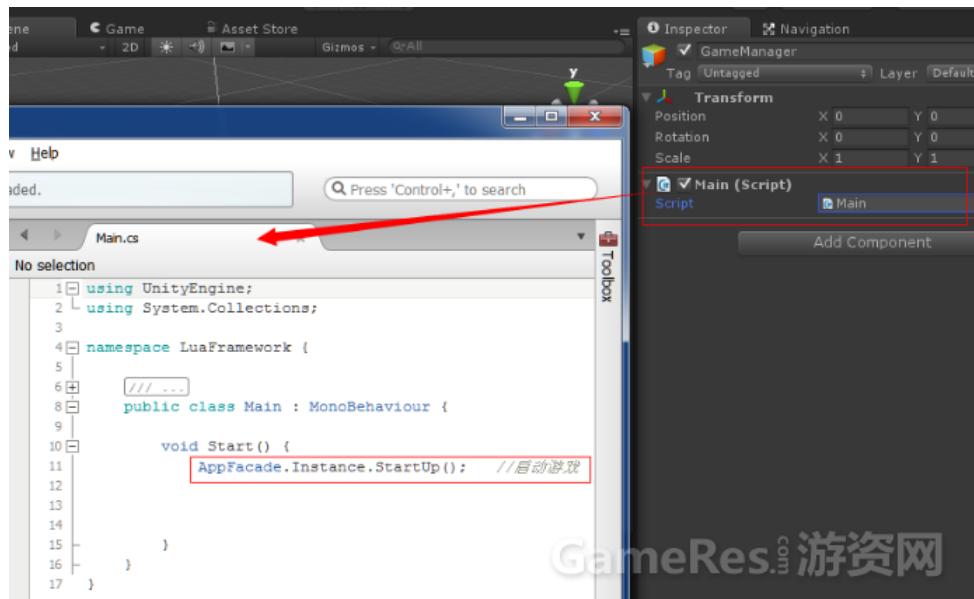
2、运行Lua代码

这一步的目标很简单，就是让框架运行我们自己写的lua代码，显示一句“helloWorld”。下一步再考虑代码的热更新问题。



1) 新建场景，并在任意物体中添加Main组件。其实Main组件里面只是调用了AppFacade.Instance.StartUp()，这是框架的起点。框架将会自动完成资源加载、热更新等等事项。

(/apps/redir
utm_source
banner-click



添加Main组件

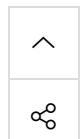
2) 删掉示例的调用。现在不需要框架自带的示例了，需要删掉一些代码，使框架只运行我们编写的lua文件。打开Assets\LuaFramework\Scripts\Manager\GameManager.cs，将OnInitialize修改成下图这个样子。这是lua的入口，框架会调用Main.lua的Main方法。

```
void OnInitialize() {
    LuaManager.InitStart();
    //LuaManager.DoFile("Logic/Game");           //加载游戏
    //LuaManager.DoFile("Logic/Network");         //加载网络
    //NetManager.OnInit();                      //初始化网络
    //Util.CallMethod("Game", "OnInitOK");       //初始化完成

    initialize = true;
}
```

lua入口

3) 打开Assets\LuaFramework\Lua\main.lua，编写lua代码。这里只添加一句“LuaFramework.Util.Log("HelloWorld");”（如下所示），它的功能相当于Debug.Log("HelloWorld")。



-主入口函数。从这里开始 lua 逻辑

```
function Main()
    LuaFramework.Util.Log("HelloWorld");
end
```

GameRes游资网

(/apps/redir
utm_source
banner-click)

“LuaFramework.Util.Log("HelloWorld")”中的Util是c#里定义的类，供lua中调用。可以打开Assets\LuaFramework\Editor\CustomSettings.cs看到所有可以供lua调用的类，如下图是CustomSettings.cs的部分语句。

```
//for LuaFramework
_GT (typeof(RectTransform)),
_GT (typeof(Text)),
 $\nearrow$ 
_GT (typeof(Util)),
_GT (typeof(AppConst)),
_GT (typeof(LuaHelper)),
_GT (typeof(ByteBuffer)),
_GT (typeof(LuaBehaviour)),
 $\searrow$ 
_GT (typeof(GameManager)),
_GT (typeof(LuaManager)),
_GT (typeof(PanelManager)),
_GT (typeof(SoundManager)),
_GT (typeof(TimerManager)),
_GT (typeof(ThreadManager)),
_GT (typeof(NetworkManager)),
_GT (typeof(ResourceManager)),
```

CustomSettings.cs的部分语句

再由具体的类可以查找所有的API（参见下面两个图），如下图是Util类的部分语句。

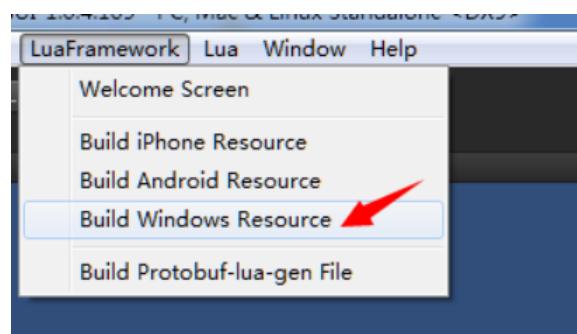
```
public static void Log(string str) {
    Debug.Log(str);
}

public static void LogWarning(string str) {
    Debug.LogWarning(str);
}

public static void.LogError(string str) {
    Debug.LogError(str);
}
```

GameRes游资网

4) 运行游戏。点击菜单栏中LuaFramework→Build Windows Resource，生成资源文件。然后运行游戏，即可在控制台中看到打印出的HelloWorld。



生成资源文件

Project
Console
Clear
Collapse
Clear on Play
Error Pause

```

UnityEngine.Debug.Log(Object)
11:28:17.91-78: [string "tolua.lua"]:9: jit opt to 3
UnityEngine.Debug.Log(Object)
11:28:17.96-78: [string "tolua.lua"]:11: jit true CMOV SSE2 SSE3 SSE4.1 fold cse dce fwd
UnityEngine.Debug.Log(Object)
11:28:17.101-78: [string "tolua.lua"]:12: os: Windows, arch: x64
UnityEngine.Debug.Log(Object)
HelloWorld
UnityEngine.Debug.Log(Object)

```

(/apps/redir
utm_source
banner-click)

GameRes.com 游资网

运行结果

5) 调试模式。按照默认的设置，每更改一次lua代码，都需要执行Build XXX Resource 才能生效。读者可以将Assets\LuaFramework\Scripts\ConstDefine\AppConst.cs中的Lua BundleMode修改为false，这样代码文件便不会以AssetBundle模式读取，会直接生效，以方便调试。

AppConst.cs
X
AppConst
No selection

```

16     /// 如果开启更新模式，前提必须启动框架自带服务器端。
17     /// 否则就需要自己将StreamingAssets里面的所有内容
18     /// 复制到自己的Webserver上面，并修改下面的WebUrl。
19     /// </summary>
20     public const bool UpdateMode = false;
21     public const bool LuaByteMode = false;
22     public const bool LuaBundleMode = false;

```

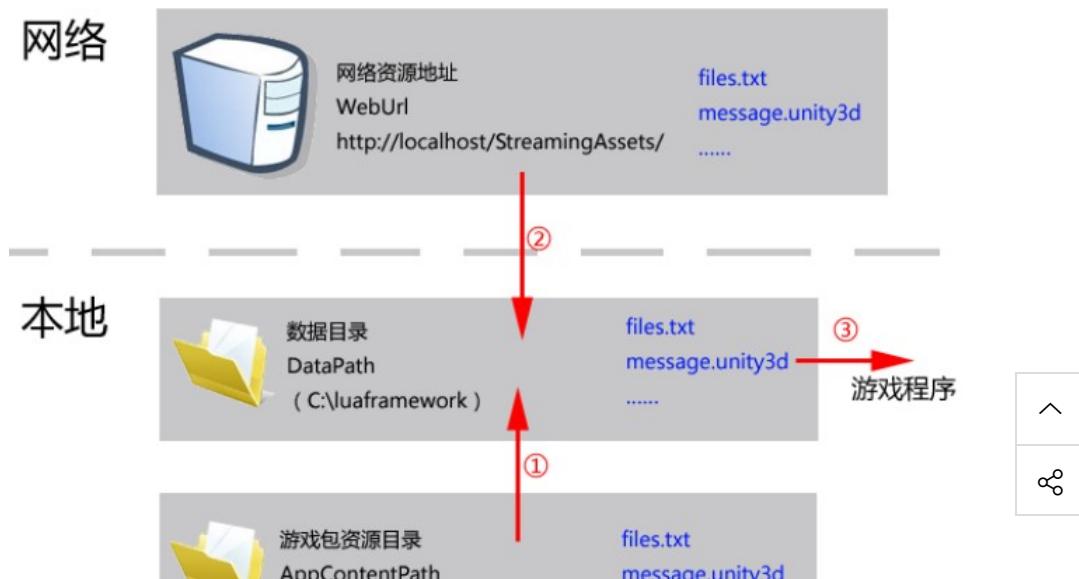
//更新模式-默认关闭
//Lua字节码模式-默认关闭
//Lua/AssetBundle模式

设置LuaBundleMode

3、代码热更新

热更新的原理

接下来便要尝试代码热更新，让程序下载服务器上的lua文件，然后运行它。在说明热更新之前，需要先看看Unity3D热更新的一般方法。如下图所示，Unity3D的热更新会涉及3个目录。



热更新的过程

游戏资源目录：里面包含Unity3D工程中StreamingAssets文件夹下的文件。安装游戏之后，这些文件将会被一字不差地复制到目标机器上的特定文件夹里，不同平台的文件夹不同，如下所示（上图以windows平台为例）

```
Mac OS 或 Windows: Application.dataPath + "/StreamingAssets";
IOS: Application.dataPath + "/Raw";
Android: jar:file://" + Application.dataPath + "!/assets/";
```

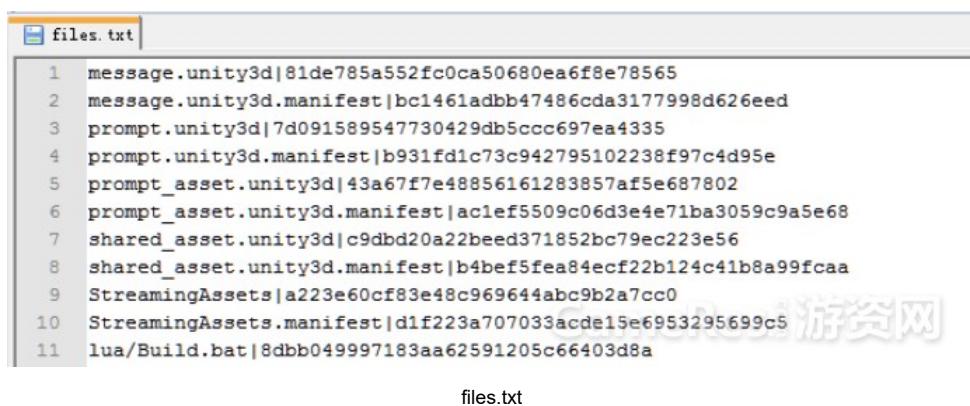
数据目录：由于“游戏资源目录”在Android和IOS上是只读的，不能把网上的下载的资源放到里面，所以需要建立一个“数据目录”，该目录可读可写。第一次开启游戏后，程序将“游戏资源目录”的内容复制到“数据目录中”（步骤1，这个步骤只会执行一次，下次再打开游戏就不复制了）。游戏过程中的资源加载，都是从“数据目录”中获取、解包（步骤3）。不同平台下，“数据目录”的地址也不同，LuaFramework的定义如下：

```
Android 或 IOS: Application.persistentDataPath + "/LuaFramework"
Mac OS 或 Windows: c:/LuaFramework/
调试模式下: Application.dataPath + "/StreamingAssets/"
```

注：“LuaFramework”和“StreamingAssets”由配置决定，这里取默认值

网络资源地址：存放游戏资源的网址，游戏开启后，程序会从网络资源地址下载一些更新的文件到数据目录。

这些目录包含着不同版本的资源文件，以及用于版本控制的files.txt。Files.txt的内容如下图所示，里面存放着资源文件的名称和md5码。程序会先下载“网络资源地址”上的files.txt，然后与“数据目录”中文件的md5码做比较，更新有变化的文件（步骤2）。



```
files.txt
1 message.unity3d|81de785a552fc0ca50680ea6f8e78565
2 message.unity3d.manifest|bc1461adb47486cda3177998d626eed
3 prompt.unity3d|7d091589547730429db5ccc697ea4335
4 prompt.unity3d.manifest|b931fd1c73c942795102238f97c4d95e
5 prompt_asset.unity3d|43a67f7e48856161283857af5e687802
6 prompt_asset.unity3d.manifest|ac1ef5509c06d3e4e71ba3059c9a5e68
7 shared_asset.unity3d|c9dbd20a22beed371852bc79ec223e56
8 shared_asset.unity3d.manifest|b4bef5fea84ecf22b124c41b8a99fc当地
9 StreamingAssets|a223e60cf83e48c969644abc9b2a7cc0
10 StreamingAssets.manifest|d1f223a707033acde15e6953295699c5
11 lua/Build.bat|8dbb049997183aa62591205c66403d8a
```

files.txt

LuaFramework的热更新代码定义在Assets\LuaFramework\Scripts\Manager\GameManager.cs，真正用到项目时可能还需少许改动。

开始热更新代码吧！

(/apps/redir
utm_source
banner-click)

那么开始测试热更新代码的功能吧！热更上述实现的“HelloWorld”。

1) 修改配置。框架的默认配置是从本地加载文件，需要打开AppConst.cs将UpdateMode设置为true（才会执行步骤2），将LuaBundleMode设置为true，将WebUrl设置成服务器地址。如下图所示。

(/apps/redir
utm_source
banner-click

```
namespace LuaFramework {
    public class AppConst {
        public const bool DebugMode = false; // 测试模式-用于内部测试
        /// <summary>
        /// 如果想删掉框架自带的例子，那这个例子模式必须要
        /// 关闭，否则会出现一些错误。
        /// </summary>
        public const bool ExampleMode = true; // 例子模式

        /// <summary>
        /// 如果开启更新模式，前提必须启动框架自带服务器端。
        /// 否则就需要自己得StreamingAssets里面的所有内容
        /// 复制到自己的Webserver上面，并修改下面的WebUrl。
        /// </summary>
        public const bool UpdateMode = true; // 更新模式-默认关闭
        public const bool LuaByteMode = false; // Lua字节码模式-默认关闭
        public const bool LuaBundleMode = true; // Lua代码AssetBundle模式

        public const int TimerInterval = 1; // 游戏帧率
        public const int GameFrameRate = 30;

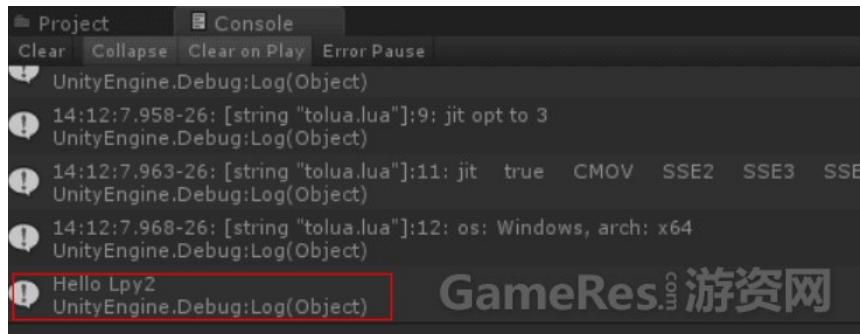
        public const stringAppName = "LuaFramework"; // 应用程序名称
        public const stringLuaTempDir = "Lua/"; // 暂时目录
        public const stringAppPrefix = AppName + "_"; // 应用程序前缀
        public const stringExtName = ".unity3d"; // 素材扩展名
        public const stringAssetDir = "StreamingAssets"; // 素材目录
        public const stringWebUrl = "http://localhost/StreamingAssets/"; // 测试更新地址

        public static stringUserId = string.Empty; // 用户ID
        public static intSocketPort = 0; // Socket服务器端口
        public static stringSocketAddress = string.Empty; // Socket服务器地址
    }
}
```

AppConst的配置

2) 配置“网络资源”。笔者使用IIS开启本地服务器，然后将StreamingAssets里面的所有内容复制到服务器上面。必要时要配置一些权限，让所有文件都都可以下载

3) 测试热更。改一下Lua脚本（如将HelloWorld改为Hello Lpy2），点击Build Windows Resource，将“工程目录/StreamingAssets”里面的文件复制到服务器上。再将脚本改成其他内容，然后Build Windows Resource，覆盖掉本地资源。运行游戏，如果程序显示“Hello Lpy2”的代码，证明成功从网上拉取了文件。



代码热更新



附录：

一、语法：

局部变量声明:用local声明 相当于javascript var

(/apps/redir
utm_source
banner-click)

声明类: classA={}

int类型转换成string:

tostring(i);

字符串相加+用“..”: "A".."B"="AB"

属性获取用. 调用方法用: A.a; A:a();

if条件语句: if a==0 then print('干嘛') end

if..else条件控制: if 条件 then ... elseif 条件 then ... else ... end

注意:lua 中没有 switch case 语句

1.应用类库using UnityEngine: luanet.load_assembly('UnityEngine')

2.lua全局使用C# Coment , define.lua声明后其他lua脚本直接使用:

GameObject=UnityEngine.GameObject

ParticleSystem=UnityEngine.ParticleSystem

3.添加脚本: newGameObj.AddComponent(luanet.ctype(ParticleSystem))

4.场景新建一个GameObject: local newGameObj=GameObject('NewObj')

5.Unity对象转lua对象:

localGo (<https://link.jianshu.com?t=http://lib.csdn.net/base/go>)= newObject(obj);

6.C#获得Lua函数LuaFunction f=l.GetFunction("函数名"); f.Call(参数)调用

7.lua协程开启: coroutine.start(方法名); 等待: coroutine.wait(时间s);

8.根据transform查找子对象:

local label = go.transform:FindChild('##/Text');

9.获取脚本: transform:GetComponent('LuaBehaviour');

10.输出信息:

logWarn("OnDestroy---->>>");

log(go.name);



一、tolua#

c#调用lua: LuaState[变量名/函数名]

1.LuaState

a.执行lua代码段

DoString(string)

DoFile(.lua文件名)

Require(.lua文件名(但没有.lua后缀))

b.获取lua函数或者表

LuaFunction func = lua.GetFunction(函数名); 或者 LuaFunction func = lua[函数名] as LuaFunction;

LuaTable table = lua.GetTable(表名);

c.Start(): 如果需要使用wrap, 则需要调用该方法

2.LuaFunction

Call()

3.LuaTable

LuaTable[变量名/函数名]

ToArray()

lua调用c#

在c#中将引用传递到lua中后:

1.通过“.”(点号)来使用非静态的变量以及静态的变量与方法

2.通过“:”(冒号)来使用非静态的方法

3.通过“{}”来传递数组给c#

4.创建GameObject: newObject(变量)

5.摧毁GameObject: destroy(变量)

6.获取组件: GetComponent('LuaBehaviour')

二、LuaFramework(使用PureMVC)

a.基础模块

(/apps/redir
utm_source
banner-click



1.Util：对Mono的功能进行封装，这样不继承Mono的类就能使用Mono的东西了(如transform.Find、GetComponent)；还有其他的工具方法

2.AppFacade：继承Facade，整套框架的入口

(/apps/redir
utm_source
banner-click)

3.Base：继承MonoBehaviour，是一切View和Manager的基类；持有各种Manager的引用；能注册移除(view所感兴趣的)信息

4.View：只有一个方法：public virtual void OnMessage(IMessage message) 这是处理信息的方法

5.Manager：继承Base

6.AppView：继承View，是一个范例：注册View所感兴趣的信息，处理信息

b.lua模块

1.LuaFileUtils：通过.lua文件路径和AssetBundle文件路径这两种方式来找.lua文件，并读取返回byte[]

2.LuaLoader：继承LuaFileUtils，并无重要变化

3.LuaEvent：类似c#中的event，提供Add和Remove方法

4.LuaLooper：继承MonoBehaviour，在Update / LateUpdate / FixedUpdate中执行对应的LuaEvent

5.LuaBinder：如果执行的.lua文件需要用到unity中的类型，则需要用这个类给LuaState进行绑定

6.LuaManager：继承Manager，入口类，初始化Lua代码加载路径(调试模式下是在Assets\LuaFramework目录下，非调试模式是在C:\luaframework\lua(window系统)，默认是非调试模式)，引用一个LuaState并封装其功能(读取lua文件、调用方法等)

7.LuaBehaviour：继承View，在Awake / Start中调用lua中对应的方法；并提供点击事件的相关处理

c.Manager模块

1.ResourceManager：加载AssetBundle的相关操作。在pc平台上默认加载的是Assets\StreamingAssets里的东西，移动平台上则是Application.persistentDataPath。那么如果我们想在pc平台上像移动平台一样读取外部路径(使用www)，在pc平台模拟热更新，那么就可以在Initialize这个方法修改：m_BaseDownloadingURL = "file://" + Util.DataPath;

例子1

展示了最小的tolua#环境，以及执行一段lua代码操作代码如下：

...

```
LuaStatelua=newLuaState();lua.Start();stringhello=      @"print('hello tolua#')";lua.D  
oString(hello, "HelloWorld.cs");lua.CheckTop();lua.Dispose();lua=null;
```



...

LuaState封装了对lua 主要数据结构 lua_State 指针的各种堆栈操作。

一般对于客户端，推荐只创建一个LuaState对象。如果要使用多State需要在Unity中设置全局宏 MULTI_STATE

(/apps/redir
utm_source
banner-click)

LuaState.Start 需要在tolua代码加载到内存后调用。如果使用assetbundle加载lua文件，调用Start()之前assetbundle必须加载好

LuaState.DoString 执行一段lua代码,除了例子,比较少用这种方式加载代码,无法避免代码重复加载覆盖等,需调用者自己保证。第二个参数用于调试信息,或者error消息(用于提示出错代码所在文件名称)

LuaState.CheckTop 检查是否堆栈是否平衡, 一般放于update中, c#中任何使用lua堆栈操作, 都需要调用者自己平衡堆栈 (参考LuaFunction以及LuaTable代码), 当CheckTop出现警告时其实早已经离开了堆栈操作范围, 这是需自行review代码。

LuaState.Dispose 释放LuaState 以及其资源。

注意:此例子无法发布到手机

例子2 (<https://link.jianshu.com?t=https://github.com/topameng/tolua/blob/master/Assets/ToLua/Examples/README.md#%E4%BE%8B%E5%AD%902>)

展示了dofile跟require的区别, 代码如下:

...

```
LuaStatelua=null;voidStart() {     lua =newLuaState();         lua.Start();//  
如果移动了ToLua目录, 需要自己手动, 这里只是例子就不做配置了stringfullPath= Application.dataPath +"/ToLua/Examples/02_ScriptsFromFile";     lua.AddSearchPath(fullPath); }voidOnGUI() {if(GUI.Button(newRect(50,50,120,45),"DoFile")) {  
    lua.DoFile("ScriptsFromFile.lua"); }elseif(GUI.Button(newRect(50,150,120,45),"Require")) {  
    lua.Require("ScriptsFromFile"); }    lua.Collect();    lua.CheckTop(); }voidOnApplicationQuit() {    lua.Dispose();    lua=null; }
```

...

tolua#DoFile函数,跟lua保持一致行为,能多次执行一个文件。tolua#加入了新的Require函数,无论c#和lua谁先require一个lua文件,都能保证加载唯一性

LuaState.AddSearchPath 增加搜索目录, 这样DoFile跟Require函数可以只用文件名,无需写全路径

LuaState.DoFile 加载一个lua文件, 注意dofile需要扩展名, 可反复执行, 后面的变量会覆盖之前的DoFile加载的变量

LuaState.Require 同lua require(modname)操作, 加载指定模块并且把结果写入到package.loaded中,如果modname存在, 则直接返回package.loaded[modname]



`LuaState.Collect` 垃圾回收, 对于被自动gc的LuaFunction, LuaTable, 以及委托减掉的Lua Function, 延迟删除的Object之类。等等需要延迟处理的回收, 都在这里自动执行

注意:虽然有文件加载,但此例子无法发布到手机, 如果ToLua目录不在/Assets目录下, 需要修改代码中的目录位置

(/apps/redir
utm_source
banner-click)

例子3 LuaFunction (<https://link.jianshu.com?t=https://github.com/topameng/tolua/blob/master/Assets/ToLua/Examples/README.md#%E4%BE%8B%E5%AD%903-luafunction>)

展示了如何调用lua的函数, 主要代码如下:

...

```
private string script=      @"function luaFunc(num) return num + 1 end test = {} test.luaFunc = luaFunc"; LuaFunction luaFunc=null; LuaState lua=null; void Start() { newLuaResLoader();    lua = newLuaState();    lua.Start();    DelegateFactory.Init();    lua.DoString(script); //Get the function object luaFunc = lua.GetFunction("test.luaFunc"); if(func != null) { int num= luaFunc.Invoke(123456);    Debugger.Log("generic call return: {0}", num);    num = CallFunc();    Debugger.Log("expansion call return: {0}", num);    FuncFunc= luaFunc.ToDelegate>();    num = Func(123456);    Debugger.Log("Delegate call return: {0}", num);    num = lua.Invoke("test.luaFunc",123456,true);    Debugger.Log("luastate call return: {0}", num); }    lua.CheckTop(); } void OnDestroy() { if(luaFunc != null) {    luaFunc.Dispose();    luaFunc = null; }    lua.Dispose();    lua = null; } int CallFunc() {    luaFunc.BeginPCall();    luaFunc.Push(123456);    luaFunc.PCall();    int num= (int)luaFunc.CheckNumber();    luaFunc.EndPCall();    return num; }
```

...

tolua# 简化了lua函数的操作, 通过LuaFunction封装(并缓存)一个lua函数, 并提供各种操作, 建议频繁调用函数使用无GC方式。

`LuaState.GetLuaFunction` 获取并缓存一个lua函数, 此函数支持串式操作, 如"test.luaFunc"代表test表中的luaFunc函数。

`LuaState.Invoke` 临时调用一个lua function并返回一个值, 这个操作并不缓存lua function, 适合频率非常低的函数调用。

`LuaFunction.Call()` 不需要返回值的函数调用操作

`LuaFunction.Invoke()` 有一个返回值的函数调用操作

`LuaFunction.BeginPCall()` 开始函数调用

`LuaFunction.Push()` 压入函数调用需要的参数, 通过众多的重载函数来解决参数转换gc问题

`LuaFunction.PCall()` 调用lua函数

`LuaFunction.CheckNumber()` 提取函数返回值, 并检查返回值为lua number类型



LuaFunction.EndPCall() 结束lua函数调用, 清楚函数调用造成的堆栈变化

LuaFunction.Dispose() 释放LuaFunction, 递减引用计数, 如果引用计数为0, 则从_R表删除该函数

注意:无论Call还是PCall只相当于lua中的函数'.'调用。

请注意'.'这种语法糖 self:call(...) == self.call(self, ...)

c# 中需要按后面方式调用, 即必须主动传入第一个参数self

例子4 (<https://link.jianshu.com?t=https://github.com/topameng/tolua/blob/master/Assets/ToLua/Examples/README.md>)

展示了如何访问lua中变量, table的操作

...

```
private string script=      @"print('Obj2Spawn is: '..Obj2Spawn)var2read = 42varTable = {1,2,3,4,5}varTable.default = 1varTable.map = {}varTable.map.name = 'map'meta = {name = 'meta'}setmetatable(varTable, meta)function TestFunc(strs)print('get func by variable')end";void Start(){newLuaResLoader();LuaState lua=newLuaState();    lua.Start();    lua["Obj2Spawn"] = 5;    lua.DoString(script);//通过LuaState访问Debugger.Log("Read var from lua: {0}", lua["var2read"]);    Debugger.Log("Read table var from lua: {0}", lua["varTable.default"]);//LuaState 拆串式tableLuaFunction func= lua["TestFunc"] asLuaFunction;    func.Call();    func.Dispose();//cache成LuaTable进行访问LuaTable table= lua.GetTable("varTable");    Debugger.Log("Read varTable from lua, default: {0} name: {1}", table["default"], table["map.name"]);    table["map.name"] = "new";//table 字符串只能是keyDebugger.Log("Modify varTable name: {0}", table["map.name"]);    table.AddTable("newmap");LuaTable table1= (LuaTable)table["newmap"];    table1["name"] = "table1";    Debugger.Log("varTable.newmap name: {0}", table1["name"]);    table1.Dispose();    table1 = table.GetMetaTable();if(table1 != null)    {        Debugger.Log("varTable metatable name: {0}", table1["name"]);        object[] list= table.ToArray();for(int i=0; i < list.Length; i++)        {            Debugger.Log("varTable[{0}], is {1}", i, list[i]);        }    }    table.Dispose();    lua.CheckTop();    lua.Dispose();}
```

...

luaState["Obj2Spawn"] LuaState通过重载this操作符, 访问lua _G表中的变量Obj2Spawn

LuaState.GetTable 从lua中获取一个lua table, 可以串式访问比如lua.GetTable("varTable.map.name") 等于 varTable->map->name

LuaTable 支持this操作符, 但此this不支持串式访问。比如table["map.name"] "map.name" 只是一个key, 不是table->map->name

LuaTable.GetMetaTable() 可以获取当前table的metatable

LuaTable.ToArray() 获取数组表中的所有对象存入到object[]表中

(/apps/redir
utm_source
banner-click



`LuaTable.AddTable(name)` 在当前的table表中添加一个名字为name的表

`LuaTable.GetTable(key)` 获取t[key]值到c#, 类似于 `lua_gettable`

`LuaTable.SetTable(key, value)` 等价于`t[k] = v`的操作, 类似于`lua_settable`

`LuaTable.RawGet(key)` 获取t[key]值到c#, 类似于 `lua_rawget`

`LuaTable.RawSet(key, value)` 等价于`t[k] = v`的操作, 类似于`lua_rawset`

例子5 协同一 (<https://link.jianshu.com?t=https://github.com/topameng/tolua/blob/master/Assets/ToLua/Examples/README.md#%E4%BE%8B%E5%AD%905-%E5%8D%8F%E5%90%8C%E4%B8%80>)

展示了如何使用lua协同, lua 代码如下:

...

```
function fib(n)local a, b=0,1while n>0do a, b=a+b, a+bn=n-1endreturn aendfunction CoFunc()print('Coroutine started')for i=1,10,1doprint(fib(i))coroutine.wait(0.1)endprint("current frameCount:"..Time.frameCount)coroutine.step()print("yield frameCount:"..Time.frameCount)local www=UnityEngine.WWW("http://www.baidu.com")coroutine.www(www)locals=toluatolstring(www.bytes)print(s:sub(1,128))print('Coroutine ended')endfunction TestContinue()coroutine.start(CoFunc)endlocal coDelay=nilfunction Delay()local c=1while true do coroutine.wait(1)print("Count:"..c)c=c+1endendfunction StartDelay()coDelay=coroutine.start(Delay)endfunction StopDelay()coroutine.stop(coDelay)end
```

...

c#代码如下:

...

```
newLuaResLoader();lua= newLuaState();lua.Start();LuaBinder.Bind(lua);DelegateFactory.Init();looper= gameObject.AddComponent();looper.luaState= lua;lua.DoString(luaFile.text, "TestLuaCoroutine.lua");LuaFunction f=lua.GetFunction("TestContinue");f.Call();f.Dispose();f= null;
```

...

必须启动LuaLooper驱动协同, 这里将一个lua的半双工协同装换为类似unity的全双工协同

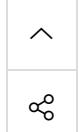
`fib`函数负责计算一个斐那波契

`coroutine.start` 启动一个lua协同

`coroutine.wait` 协同中等待一段时间, 单位:秒

`coroutine.step` 协同中等待一帧.

(/apps/redir
utm_source
banner-click



coroutine.www 等待一个WWW完成.

`toluatolstring` 转换byte数组为lua字符串缓冲

`coroutine.stop` 停止一个正在lua将要执行的协同

(/apps/redir
utm_source
banner-click

总结

1. 对于一个panel，需要添加或修改的文件：

a.添加xxxPanel & xxxCtrl

b.修改define、Game、CtrlManager

详细的可见：<http://blog.csdn.NET> (<https://link.jianshu.com?t=http://lib.csdn.net/base/donetnet/adambieber/article/details/47402805>)

2. 在lua中使用AB包内的资源的两种方法：

```
a.panelMgr>CreatePanel('Prompt', this.OnCreate);
```

```
b.resMgr:LoadPrefab('prompt', { 'PromptItem' }, this.InitPanel);
```

其中a是对b的进一步封装，因此两者都需要提供AB包名、要访问的包内资源名字(如果是panel，则默认资源名为AB包名+"Panel")以及回调方法(参数是AB包中的资源)

3.热更新的四个步骤：打包、解包、更新和加载。而这四个步骤框架已经给我们封装好了，基本上就不需要我们去管了，但还是很有必要理解其中的过程。

a. 打包：将资源全部打包到StreamingAssets文件夹

打包类: LuaFramework / Editor / Packager

打包lua文件：HandleLuaBundle，对Assets\LuaFramework\Lua 与 Assets\LuaFramework\ToLua\Lua这两个目录下的所有lua文件进行打包

打包图片等资源：HandleExampleBundle

b.解包：在移动端StreamingAssets这个文件夹是只读的，但是要做热跟新的话，就需要写入文件，因此Application.persistentDataPath这个可读可写的路径才是数据在移动端的存放路径，同时也为了比较MD5的值，就需要将StreamingAssets的东西解包(复制)到Application.persistentDataPath

c.更新：files.txt这个文件记录了所有的资源文件及其MD5值，每次进入游戏时都会从服务器下载最新的files.txt，然后对其遍历比较MD5值，如果值不同或者不存在则下载

d. 加载：先加载资源的依赖，再加载资源



那么，如果我们对外发布了一个版本1.1，然后更改资源，发布1.2，要做的就是：重新生成apk并上传，然后将StreamingAssets文件夹下的东西上传到服务器，具体位置见AppConst.WebUrl；对于用户来说，如果他安装的是1.1，那么就会下载更新，如果他安装的是1.2，那么解包之后就得到最新的资源了，无需更新了。

(/apps/redir
utm_source
banner-click

4. 整套框架的工作流程：

c#打包好后，启动游戏，GameManager会进行一些判断，如果这是游戏安装之后的第一次启动，那么就会进行解包操作。如果AppConst.UpdateMode为false，那么就不会检测更新，否则就会进行更新操作。然后进入初始化操作，调用Game.lua中的OnInitOK方法，进入lua逻辑。

lua然后调用指定控制器的Awake方法、PanelManager的CreatePanel方法，调用c#代码，创建panel，为其添加LuaBehaviour，调用xxxPanel.lua的方法，获取控件引用，进行逻辑处理。

外部学习链接：

1. Unity3d: UI面板管理整合进ToLua (<https://link.jianshu.com?t=http://www.cnblogs.com/joeshifu/p/56555064.html>)

2. Unity3D热更新LuaFramework入门实战(1-10)——代码热更新 (<https://link.jianshu.com?t=https://zhuanlan.zhihu.com/pyluo>)

3. ToLua例子脚本解析 (<https://link.jianshu.com?t=https://github.com/topameng/tolua/blob/master/Assets/ToLua/Examples/README.md>)

4. Unity5 + kbengine + ULUA(toLua) 一个KBE的Lua热更新客户端demo (<https://link.jianshu.com?t=https://github.com/liuxq/StriveGame>)

5. [Unity热更新]tolua# & LuaFramework(一～十五) (<https://link.jianshu.com?t=http://blog.csdn.net/lyh916>)

小礼物走一走，来简书关注我

赞赏支持



日记本 (/nb/15092734)

举报文章 © 著作权归作者所有



CatherinePlans (/u/f796a54054d1) ♂
写了 931819 字，被 922 人关注，获得了 1184 个喜欢
(/u/f796a54054d1)

+ 关注

喜欢 | 34



更多分享

