目录

**[一、 工程安装及操作](#_Toc23174_WPSOffice_Level1)** **[2](#_Toc23174_WPSOffice_Level1)**

[1、 下载](#_Toc16691_WPSOffice_Level2) [2](#_Toc16691_WPSOffice_Level2)

[2、 安装](#_Toc31057_WPSOffice_Level2) [2](#_Toc31057_WPSOffice_Level2)

[3、 配置](#_Toc16565_WPSOffice_Level2) [2](#_Toc16565_WPSOffice_Level2)

[4、 基本操作](#_Toc29694_WPSOffice_Level2) [2](#_Toc29694_WPSOffice_Level2)

**[二、 原理介绍](#_Toc16691_WPSOffice_Level1)** **[2](#_Toc16691_WPSOffice_Level1)**

[1、 Lua热更原理](#_Toc29621_WPSOffice_Level2) [2](#_Toc29621_WPSOffice_Level2)

[2、 Tolua介绍](#_Toc17929_WPSOffice_Level2) [3](#_Toc17929_WPSOffice_Level2)

[3、 网络模块原理](#_Toc16398_WPSOffice_Level2) [9](#_Toc16398_WPSOffice_Level2)

[4、 Unity热更原理](#_Toc3761_WPSOffice_Level2) [9](#_Toc3761_WPSOffice_Level2)

**[三、 使用方式](#_Toc31057_WPSOffice_Level1)** **[9](#_Toc31057_WPSOffice_Level1)**

[1、 网络链接及通信](#_Toc32168_WPSOffice_Level2) [9](#_Toc32168_WPSOffice_Level2)

[2、 通信加密及解析](#_Toc11597_WPSOffice_Level2) [9](#_Toc11597_WPSOffice_Level2)

[3、 Lua中间函数生成及lua编程](#_Toc16685_WPSOffice_Level2) [9](#_Toc16685_WPSOffice_Level2)

[4、 C#脚本编程](#_Toc5075_WPSOffice_Level2) [9](#_Toc5075_WPSOffice_Level2)

**[四、 源码解读](#_Toc16565_WPSOffice_Level1)** **[9](#_Toc16565_WPSOffice_Level1)**

[1、pureMVC简述](#_Toc19710_WPSOffice_Level2) [9](#_Toc19710_WPSOffice_Level2)

[2、 工程文件结构](#_Toc14934_WPSOffice_Level2) [10](#_Toc14934_WPSOffice_Level2)

[3、Editor文件夹内容解读](#_Toc25398_WPSOffice_Level2) [10](#_Toc25398_WPSOffice_Level2)

**[五、 常见错误](#_Toc29694_WPSOffice_Level1)** **[13](#_Toc29694_WPSOffice_Level1)**

**[六、 参考资料](#_Toc29621_WPSOffice_Level1)** **[15](#_Toc29621_WPSOffice_Level1)**

1. 工程安装及操作
2. 下载

LuaFramework\_UGUI下载地址：<https://github.com/jarjin/LuaFramework_UGUI>

Tolua#下载：<https://github.com/topameng/tolua>（lua\_framework\_UGUI包含了tolua#）

1. 安装

2.1、LuaFrameWork本身是一个Unity项目，我们可以直接使用Unity打开项目

2.2、在unity工程中，点击菜单栏里面的lua,在弹出菜单中，选择“Generate ALL”

2.3、在unity工程中，点击菜单栏里面的luaFrameWork,在弹出菜单中，选择”Build Windows Resource”

执行第三步时，可能会报错。我用2018.3版本的unity报错了，直接在Editor/customsetting.cs文件里面，注释掉相关的\_GT导入即可；用2017版本的不会报错(参考常见错误)。

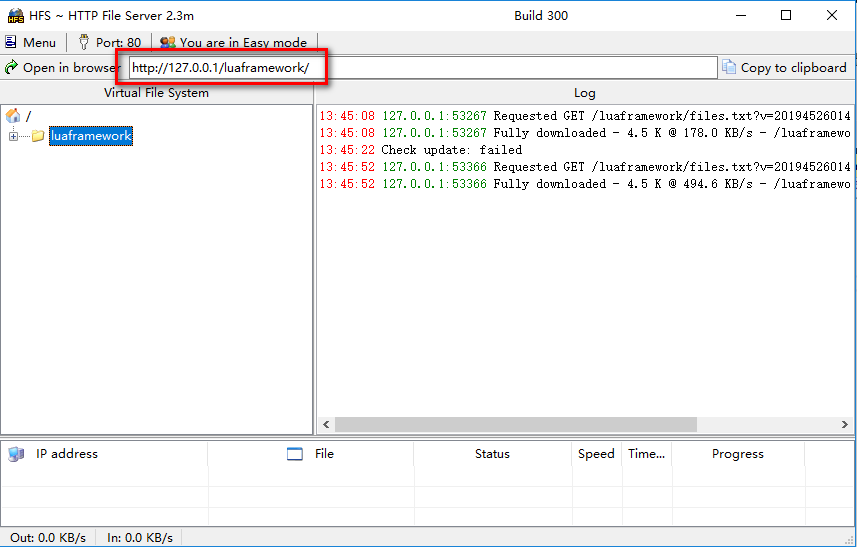
2.4、在scenes文件里面，选择场景main。点击运行，就会弹出如下界面：



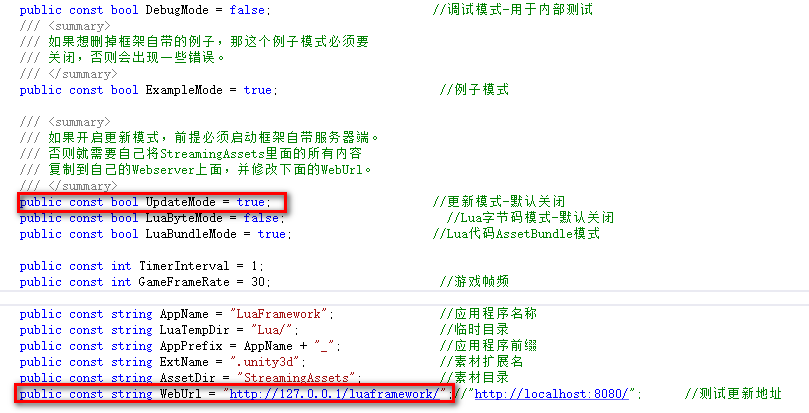
至此安装完成

1. 配置

3.1、更新模式配置：Luaframework 框架在启动的时候，第一步是检查资源是否更新到最新。（GameManager.cs中CheckExtractResource（）函数。），因此我们需要自己搭建一个简易的服务端，这里我们用hfs进行模拟。将相关文件导入其中（选择虚拟文件夹）：



另外需要注意AppConst.cs的配置，如下：



3.2、网络连接配置：

在Assets/Lua/Logic/Game.lua文件中，我们可以进行相关IP和断开的配置，当资源更新完成后，网络模块即通过该配置连接服务端。

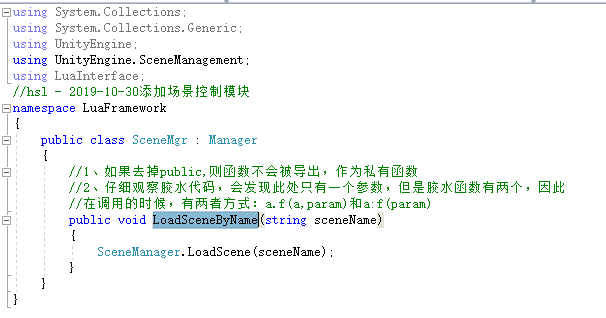


1. 基本操作

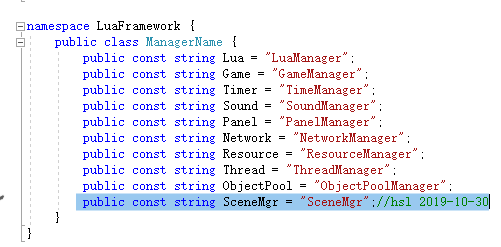
4.1、添加新的管理器模块

(如果这个地方第一次看不懂，请先跳过，可以等后面看完源码介绍，了解了相关架构知识后，再进行查阅借鉴。)Luaframework本身提供了大量管理模块,GameManager,LuaManager,ResourceManager,等，但是我们在实际使用的时候，需要在lua里面，进行场景跳转的操作，而模块刚好并未提供该模块，因此，这里我们演示如何添加一个场景的管理模块到架构中。

1. 、创建管理模块源代码。我们在Assets/LuaFramework/Scripts/Manager/文件里面创建源码文件：SceneMgr.cs;文件里面定义了SceneMgr类，一个函数LoadSceneByName（string sceneName）来装载指定的场景。



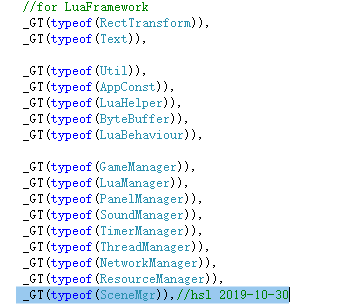
1. 、在ManagerName.cs里面，添加对应的名字字符串。



1. 、在StartUpCommand.cs文件里面，将SceneMgr加入管理器的初始化中：

AppFacade.Instance.AddManager<SceneMgr>(ManagerName.SceneMgr);

1. 、在基类中（Base.cs），添加获取模块单例的函数。详情可查看工程源代码
2. 、在LuaHelper.cs中添加场景管理器单例的获取函数，该函数最终会提供给lua脚本调用
3. 、在CustomSetting.cs添加需要导出给lua用的类。



1. 、最后，我们添加lua脚本代码，进行简单测试（点击运行例子里面的 按钮，跳转到其他场景）。

首先我们在LuaFramework\Lua\Common\define.lua中,通过函数 LuaHelper.GetSceneManager()获取sceneMgr,

然后在按钮监听函数里面（LuaFramework\Lua\Controller\PromptCtrl.lua中的OnClick()函数），添加sceneMgr:LoadSceneByName("map1")来跳转到map1场景。

执行Clear Wrap files生成胶水函数，执行Rebuild Windows Resoure生成lua的脚本资源。执行代码，即可实现测试

1. 原理介绍
2. Lua热更原理

由于游戏开发快速更替等特性，热更在游戏开发中一直处于非常重要的地位。

对于游戏服务器来说，玩家在进行网络游戏时，如果服务器程序频繁关闭，将会带来非常不友好的体验；所以当修改BUG，或者开启活动，更新玩法的时候，一般是尽量做到避免重启服务器的，此时就要求服务器进行热更。常见的热更手段是用脚本语言（lua,python）来进行功能逻辑的编写，用C或者C++来进行框架，通信等更改不频繁，性能要求高的地方使用。

对于U3D等客户端来说，一般来说玩家在第一次下载完游戏之后，就不会希望每次游戏厂商更新，都重新下载整个客户端。在运营的过程中，如游戏需要更换UI，修改逻辑，开放功能等，此时若不使用热更新技术，就需要重新打包，那么玩家也就需要通过应用商店或者其他渠道重新下载游戏。然而，热更新可以在不重新下载客户端的情况下，通过远程服务，下载最新的脚本语言（lua文件也被当做资源文件，打进在ab包中），来实现更新游戏的内容。我们经常玩的手游，一段时间后，就会远程下载一些体量非常小的资源包，当解压之后，就完成了热更。其本质就是热更的体现。

在unity中，主要的热更新方式有如下三种：

1. 使用lua编写游戏逻辑

说明：lua是一款由标准C编写的脚本语言，轻便小巧，可以与绝大多数宿主语言进行交互（C++,Java,C#等）。游戏启动时加载服务器上最新的lua字节码来执行游戏。

1. C#Light（略）
2. C#反射技术（略）

热更的基本流程:

1. 对比files清单文件
2. 更新文件
3. 解压AB包中的资源
4. 初始化

游戏运行时，从服务器下载files.txt清单文件，与本地的files.txt清单文件进行对比。如果新下载的files里面的md5值与本地files的md5值不一样，或者本地清单里面根本没有这个文件，就从服务器下载这个ab包到PersistentDataPath文件夹中。下载完毕后，解开AB包中的资源，然后完成初始化。

1. Tolua介绍

2.1、C#调用lua:

和lua的常规使用方式差不多，首先创建lua虚拟机，然后绑定相应的库，启动lua虚拟机，最 后执行相关lua文件或者指定的函数

实例代码1：

private string script = @"

function luaFunc(message)

print(message)

return 42

end

";

void Start () {

LuaState l = new LuaState();

l.DoString(script);

LuaFunction f = l.GetFunction("luaFunc");

object[] r = f.Call("I called a lua function!");

print(r[0]);

}

实例代码2：

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using LuaInterface;

public class Main : MonoBehaviour

{

private LuaState lua = null;

private LuaFunction luaFunc = null;

// Start is called before the first frame update

void Start()

{

new LuaResLoader();

lua = new LuaState();

lua.Start();

LuaBinder.Bind(lua);

lua.DoFile("Controller.lua");

CallLuaFunc("Controller.start");

}

// Update is called once per frame

void Update()

{

CallLuaFunc("Controller.update");

}

private void CallLuaFunc(string func)

{

luaFunc = lua.GetFunction(func);

luaFunc.Call();

}

}

我们知道C++是通过静态库或者动态库的形式来获取并调用LUA的API的，当我们在属主语言中，能够正确执行lua的API时，我们就可以创建lua虚拟机，来执行lua的脚本代码，也可以通过lua的API来注册C++的函数，以提供给lua进行调用。

参考链接：<https://www.cnblogs.com/slysky/p/7890286.html>

那么C#是如何获取到lua的API的呢？答案是通过DllImport来抽取相关API。

我们以tolua#来说明，在文件：Tolua\Core\LuaDLL.cs文件中，我们可以看到如下代码（部分截取）：

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

public static extern int luaL\_loadstring(IntPtr luaState, string chunk);

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

public static extern IntPtr luaL\_newstate();

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

public static extern IntPtr luaL\_gsub(IntPtr luaState, string str, string pattern, string replacement); //[-0, +1, e]

我们在\Plugins\文件夹下面，可以看到对应的lua库文件。

当C#可以调用lua的API时，那么它又是如何调用lua的函数进行执行的呢？

我们知道C++调用lua函数，其本质时调用lua的API进行执行（lua\_pcall）的，C#调用时，其本质也是如此： BeginPCall();

luaFunc.Call() 🡪 PCall(); 🡪 luaState.PCall（） 🡪lua\_pcall（）

EndPCall();

2.2、lua调用C#函数

同样我们可以回顾一下LUA是如何调用C++函数的。其实也是调用lua的API，将函数名称和函数参数类型（个数）压入lua的栈内。这个利用API将函数压入堆栈的函数我们不妨看成是一个胶水函数，利用胶水函数，这样我们就可以在lua里面进行调用。Tolua++能很好的实现这个胶水函数，可以给我们节省很多重复的工作（让我回忆起了以前用delphine的时候，每个胶水函数都要自己去写的痛苦），那么tolua#又是怎样处理的呢？让我们来分析一下：

lua源码：

local h = Test.testPrint()

print(h)

C#源码：

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Test : MonoBehaviour

{

public static string testPrint()

{

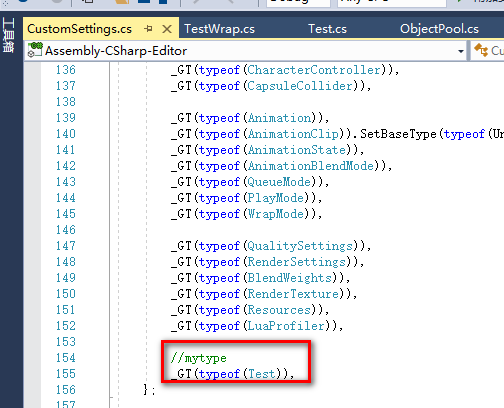
return "hello world";

}

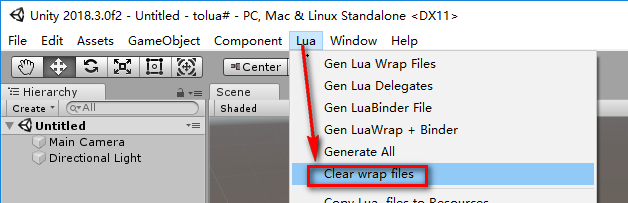
}

当我们完成代码之后，

2.2.1、我们在customSetting,cs里面添加对象



2.2.2、再运行一下clear wrap files。来重新生成胶水函数



那么让我们来看看到底发生了什么。

参考博客：https://www.cnblogs.com/blueberryzzz/blueberryzzz/p/9672342.html

当我们执行这个过程后，会发现Assets\Source\Generate目录下，多了一个文件TestWrap.cs，内容如下：

using System;

using LuaInterface;

public class TestWrap

{

public static void Register(LuaState L)

{

L.BeginClass(typeof(Test), typeof(UnityEngine.MonoBehaviour));

L.RegFunction("testPrint", testPrint);

L.RegFunction("\_\_eq", op\_Equality);

L.RegFunction("\_\_tostring", ToLua.op\_ToString);

L.EndClass();

}

[MonoPInvokeCallbackAttribute(typeof(LuaCSFunction))]

static int testPrint(IntPtr L)

{

try

{

ToLua.CheckArgsCount(L, 0);

string o = Test.testPrint();

LuaDLL.lua\_pushstring(L, o);

return 1;

}

catch (Exception e)

{

return LuaDLL.toluaL\_exception(L, e);

}

}

[MonoPInvokeCallbackAttribute(typeof(LuaCSFunction))]

static int op\_Equality(IntPtr L)

{

try

{

ToLua.CheckArgsCount(L, 2);

UnityEngine.Object arg0 = (UnityEngine.Object)ToLua.ToObject(L, 1);

UnityEngine.Object arg1 = (UnityEngine.Object)ToLua.ToObject(L, 2);

bool o = arg0 == arg1;

LuaDLL.lua\_pushboolean(L, o);

return 1;

}

catch (Exception e)

{

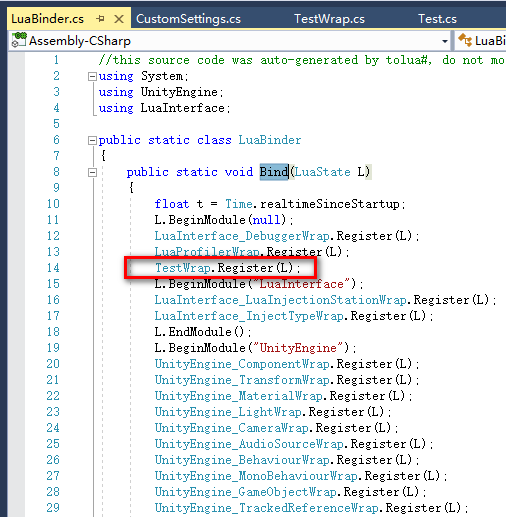
return LuaDLL.toluaL\_exception(L, e);

}

}

}

查看核心的代码我们就会发现，该函数正在通过LUA的API进行对象和函数的压栈过程。并且自动生成注册代码：



因此在我们调用LuaBinder.Bind(lua)的时候，就会注册到lua虚拟机中去。

（文件的生成过程和详细介绍，请查看前面推荐的参考博客）

至此，我们发现lua与宿主语言的相互调用其本质还是利用lua的API去进行相关操作，不一样的地方只在于宿主语言的特性和封装。万变不离其宗，如果你有时间和能力，也可以写一个tolua#,当然不要重复制作轮子，除非真的有必要。

在tolua文件里面，ToLuaExport.cs里面导出胶水文件；ToLuaMenu.cs里面响应菜单控制；ToLuaTree.cs树形结构处理的辅助文件。

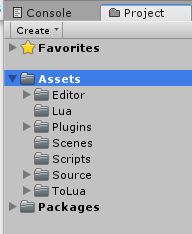
2.3、实践操作：

(1)、Tolua#下载：<https://github.com/topameng/tolua>

(2)、新创建unity3d工程

(3)、将tolua里面的Assets 与Unity5.x(根据你的unity平台而定)文件夹复制到工程目录下，覆盖掉相同的文件

(4)、在Assets目录下创建Scripts文件夹。得到目录如下：



Lua脚本放lua文件夹，CS脚本放Scripts。具体实例参考：<https://zhuanlan.zhihu.com/p/42472089>（已经推荐，B站搜索tolua,有对应的视频教程。）

1. 网络模块原理
2. Unity热更原理
3. 使用方式
4. 网络链接及通信
5. 通信加密及解析
6. Lua中间函数生成及lua编程
7. C#脚本编程
8. 源码解读

1、pureMVC简述

Luaframework 是基于pureMVC框架进行开发的。为此，我们有必要了解一下pureMVC框架，请参考给出的链接了解该模块，由于该模块并不是本文的主要内容，因此本文不做详细说明。

这里只是简单说一下pureMVC。PureMVC的核心模块由Model（数据），View（UI组件），Controller（控制）三部分组成，pureMVC官方提供了多种语言的版本，我们使用的是C#版本。在pureMVC中，M,V,C都被设置为唯一的对象（单例），并且M,V,C之间不进行直接的交互（原则上不进行，其代理可以），通过一个Facade对象来对三者进行实例化和管理（即外观模式），如此以来，用户开发时，尽可能的避免对M,V,C进行操作，而是简化为只操作Facade即可。

为了进一步减少耦合性，每一个模块又对应着自己的代理或者中介。

Model ------------->Proxy（对数据的操作）

View --------------->Mediator（对UI进行更新和监听）

Controller---------->Command（映射消息，执行其他command）

举例说明：假设这样一个场景，一个text文本，两个按钮，分别对应着加和减的功能。

则我们大抵会创建如下类：

1. AppFacade:实例化Proxy,Mediator,Command,并且注册进PMVC模块.
2. Data：剥离出text的数据，即该类里面只有一个string
3. Proxy: 该对象会对Data里面的数据进行操作，并且通知mediator
4. Mediator:更新text内容，监听Proxy传递过来的消息，并且这里面还应该监听 两个按钮的事件
5. SubCommand:减指令对应的操作，这里面可以直接访问Proxy对象，进行修改。
6. AddCommand:加指令对应的操作，这里面可以直接访问Proxy对象，进行修改。

---------------------------------------------------------------------------------------------------------------------------------

上面的都是废话，下面的文字看明白了基本就够了。

pureMVC虽然需要花时间学习，但是luaframework只使用了它很小一部分内容。只需要明白：会在appfacade中注册command,然后其他地方发生指令的时候，PMVC会调用command对应的Execute函数即可。

例如luaframework中，Assets/LuaFramework/Scripts/Framework/AppFacade.cs中，初始化的时候，代码如下：

override protected void InitFramework()

{

base.InitFramework();

RegisterCommand(NotiConst.START\_UP, typeof(StartUpCommand));

}

通过RegisterCommand(NotiConst.START\_UP, typeof(StartUpCommand));注册了StartUpCommand指令。因此当执行SendMessageCommand(NotiConst.START\_UP);的时候，PMVC会直接执行StartUpCommand对象的Execute函数。

1. 工程文件结构

2.1、Editor:

里面有两个文件customSetting.cs和Packager.cs，分别对应着菜单lua和luaframework。分别的主要功能是生成lua的胶水函数，以及资源的打包

2.2、Examples:

自带例子

2.3、Lua:

里面主要包含了一些lua相关的第三方库，和游戏功能代码，例如启动游戏，网络，协议等。我们以后的相关代码也会放在这里面。

2.4、LuaJit:

lua编译器

2.5、Resources:

项目资源存放目录

2.6、Scenes:

项目场景存放目录

2.7、Scripts:

里面包含使用CS编写的一些功能代码以及luaframework作者简化后的PureMVC源代码。

2.8、toLua:

即tolua#,用于生成lua与C#实现相互调用的胶水函数

2.9、Plugins:

框架自带的相关系统对应的插件，即一些so,a,和dll等库。

3、Editor文件夹内容解读

3.1、customSetting.cs解析：

该文件主要用于配置需要导出为胶水文件的数据类型，其中：

staticClassTypes：

导出时强制做为静态类的类型。unity 有些类作为sealed class, 其实完全等价于静态类

customDelegateList：附加导出委托类型

customTypeList：在这里添加你要导出注册到lua的类型列表

dynamicList：在这里导出动态类型

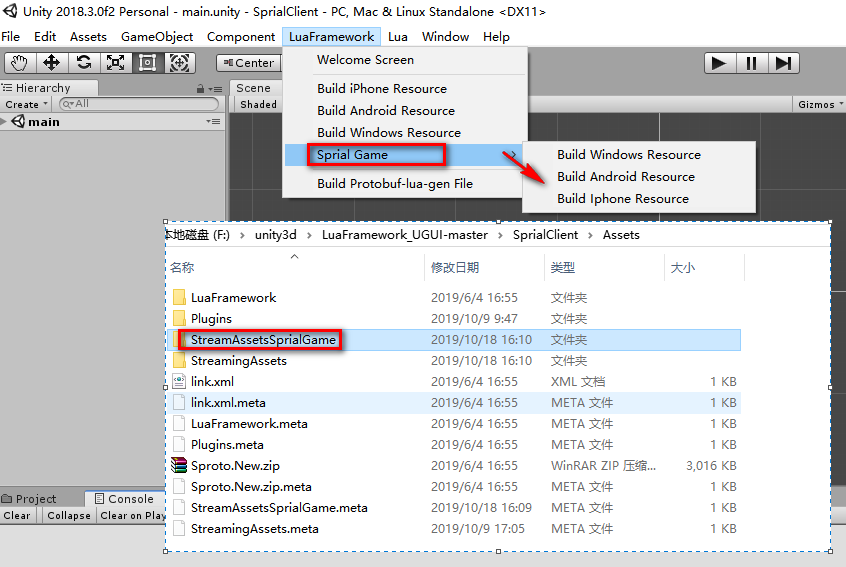
根据我们自己编写的脚本，来添加到相应的配置列表中（如果类型不清晰，可以查看已经注册进去的类型，其实特殊类型主要是Unity本身的API，而且模块差不多已经都注册了）。然后按照对应的类型，点击相应的菜单，但是安全起见，我们通常点击生成所以的胶水函数：Generate All,或者clear wrap files。

Tolua#如何生成胶水语言，详情请参考第二章“原理介绍”里面的tolua小节。

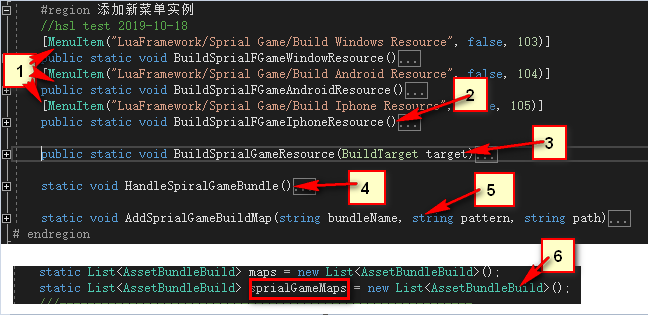
3.2、 Packager.cs解析：

该文件主要功能是打包，打包内容包括AB包，lua代码等。

其余内容可以自行查看，这里提一点需要注意的地方：一个初始化的luaframework,只能打包例子自带的资源，但是我们可能会有多个项目或者多个场景，甚至多个游戏，因此我们需要自己添加新的菜单，新的打包代码，以及新的文件夹路径。下面演示添加一个Sprial Game菜单，即这个效果：



代码实现说明如下（所有的代码都在packager.cs文件中完成）：



1. 添加菜单的代码，每一行菜单下面即对应的处理函数。
2. 菜单下面对应的处理函数。
3. 函数2调用函数3，在函数3中修改路径,且该函数为实际打包（AB包）函数
4. 函数3调用函数4，函数4访问指定文件里面指定的文件进行打包
5. 函数4调用函数5，函数5将相关文件记录插入缓存（即变量6所示的内容sprialGameMaps）中
6. Lua文件夹内容解读

3.1、Lua第三方库（略）

cjson：轻量级json解析器，sproto skynet自身的协议，类似于protobuffers。（我们最近关注一下flatbuffers,也是谷歌开发）剩下的几个不管。

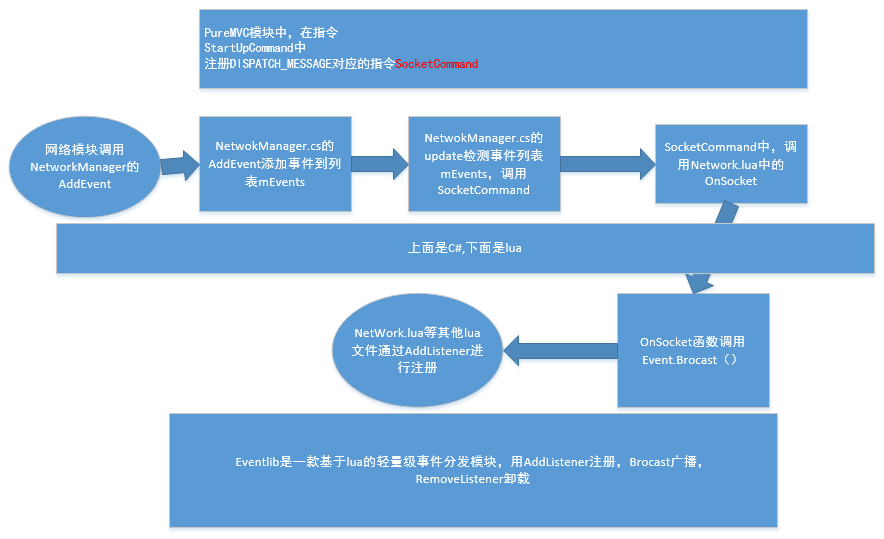
3.2、Common：

里面主要定义了一些常用的，公用的变量和函数，例如打印日志函数，常用变量，协议等，点开文件一看便知，其中，重点介绍一些协议部分。主要是说明协议是如何在代码中传递的。

点开protocal.lua文件,我们看到初始的只有4个协议，后期如果按照这种形式，服务端与客户端都会使用相同的协议，即协议号的定义完全匹配。

当开启luaframework的时候，程序执行StartUpCommand对象的Execute函数。在该函数中，我们初始化了各个管理模块，并且注册了DISPATCH\_MESSAGE指令，当发送DISPATCH\_MESSAGE指令的时候，会执行SocketCommand的Execute函数，在Execute中，执行Network.lua的OnSicket函数，进一步执行Event.Brocast。而工程文件中，只有NetworkManager.cs中的update中，会进行发送DISPATCH\_MESSAGE。update函数检查mEvents中是否存在记录，存在，即触发事件。

而lua这块，又是基于一款轻量级的事件触发模块（lua/eventlib）,通过(lua/events)Event.AddListener进行注册，Event.RemoveListener进行卸载，Event.Brocast()进行广播。如下图：



整个流程即如此。联合服务端，即：服务端通过socket，传递相应的协议号以及参数，由C#代码传递给lua,进行处理。反过来也是如此：客服端通过socket发送相应的协议号以及参数给服务端，服务端通过监听来进行处理。

3.3、Controller：

是例子对应的一些Command事件，我们本身可以不用管，它是根据pureMVC的思想来进行处理。

3.4、Logic:

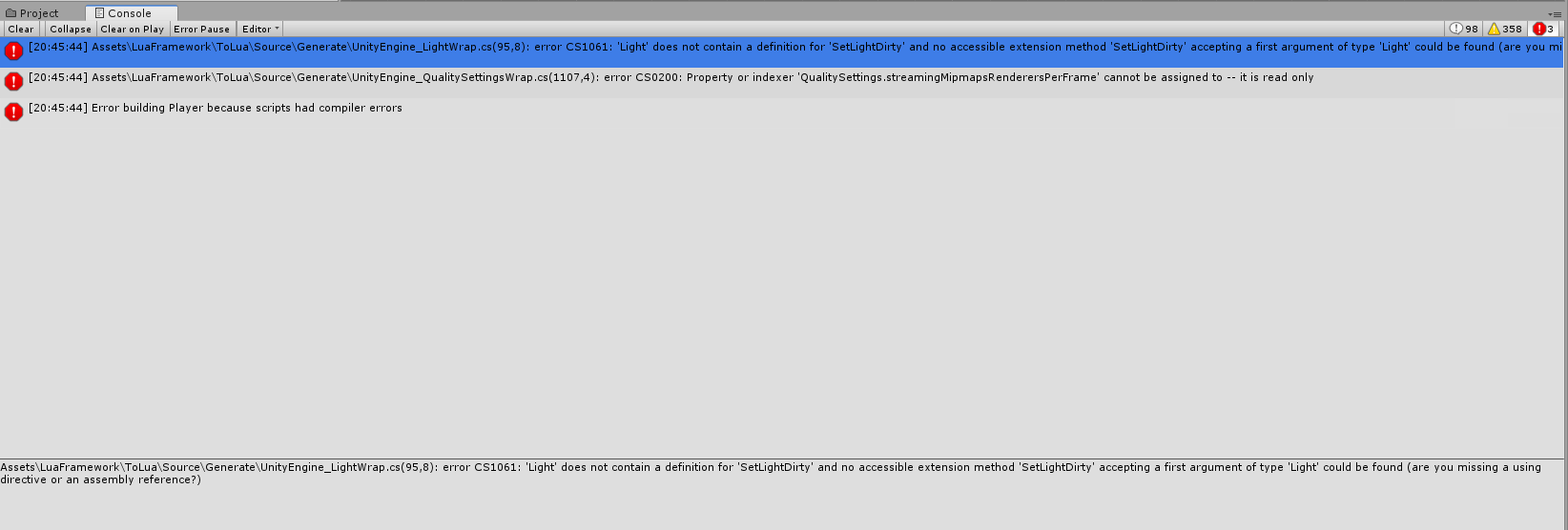
主要提供一些共C#调用的函数，game会被调用来连接网络，NetWork则执行一些网络的监听事件，其余文件暂且不管。

3.5、View:

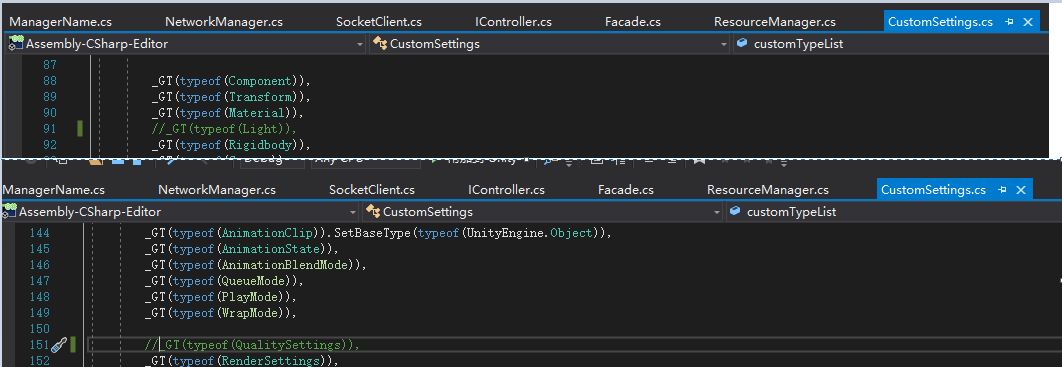
同Controller.

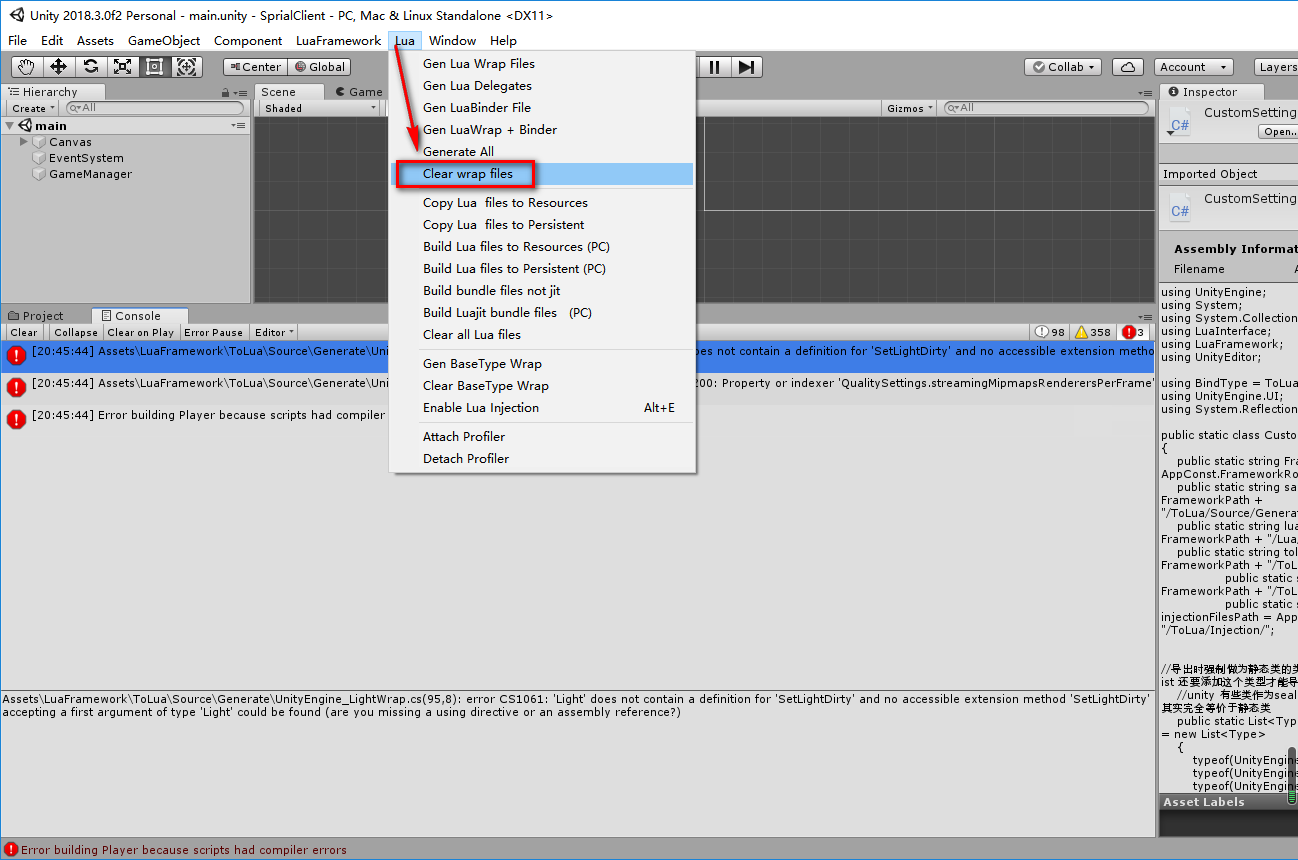
5、Scripts文件夹内容解读

1. 常见错误
2. 第一次装载luaframework，在rebuild windows resource的时候，可能会报如下错误：



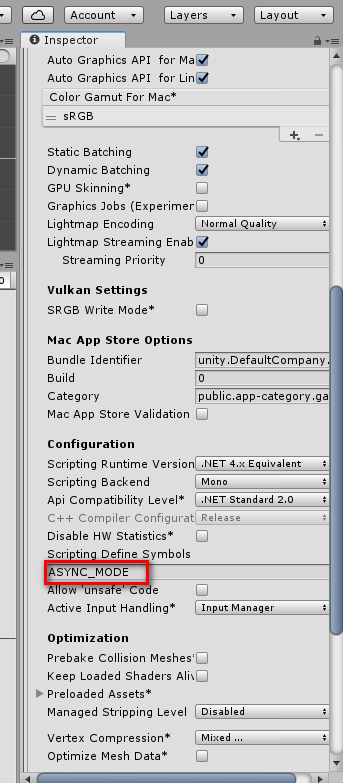
这是因为该unity版本的原因，导致luaframework兼容不了相应的变化。我们只需要注释掉配置文件Editor/customsetting.cs里面对应的内容即可。例如上面报了Light以及QualitySetting的问题，所以我们进行如下注释：



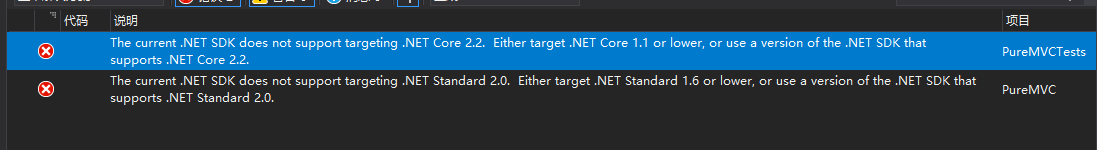
然后重新执行CLear warp files,重新执行rebuild windows resourec即可

1. 报错如下：

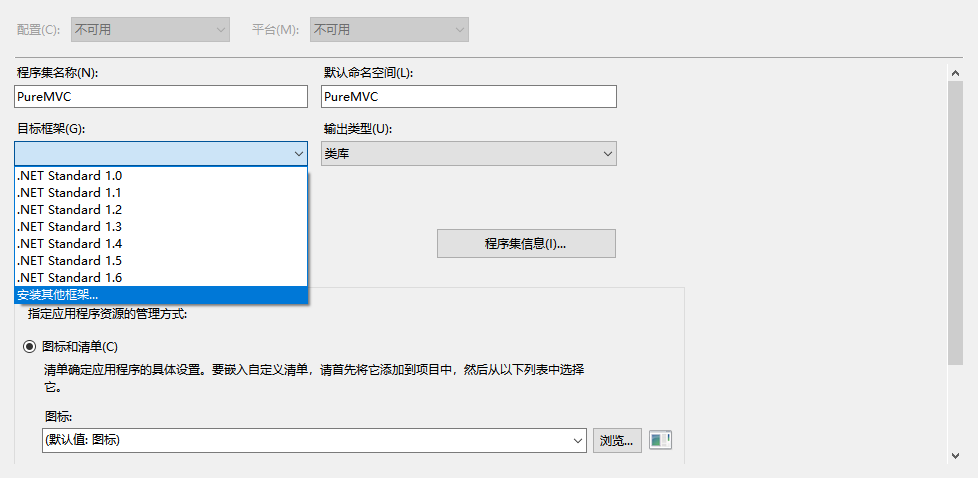
请检查是否有配置相应的参数，以上错误，需要配置ASYNC\_MODE:



1. 编译PureMVC的时候，报错：



这里我们需要安装一下.Net standard 2.2 即可：



1. 参考资料

PureMVC官网：<http://puremvc.org/>

PureMVC官网中文介绍：http://puremvc.org/docs/PureMVC\_IIBP\_Chinese.pdf

Luaframework\_UGUI官网：http://www.ulua.org/index.html

unity ToLua&LuaFramework\_UGUI学习笔记：<https://www.jianshu.com/p/ef1e2641e0c6>

Unity3d + PureMVC框架搭建:https://www.cnblogs.com/AaronBlogs/p/7056645.html

PureMVC框架在Unity中的应用：<https://gameinstitute.qq.com/community/detail/127468>

热更说明：<https://zhuanlan.zhihu.com/p/42472089>

Tolua#原理：<https://www.cnblogs.com/xiuxiu55/p/7904053.html>

Tolua#原理：<http://richbabe.top/2018/07/20/ulua-tolua原理解析/>

Unity3d&lua开发环境搭建参考资料

IntelliJ IDEA 官网:https://www.jetbrains.com/idea/

IntelliJ IDEA 激活码：<https://www.cnblogs.com/lijinran/p/10720357.html>（第二个我使用成功）

IntelliJ IDEA调试lua工程环境搭建和方法：https://blog.csdn.net/sinat\_24229853/article/details/79226608