01.热更新:

玩过手游的同学一定知道,每次游戏有更新的时候,并不需要去重新下载,只要重新登录的时候等待更新完毕后就可以进入游戏.在这个等待的过程,其实就是游戏从服务器下载新的AssetBundle文件以及LUA脚本.AssetBundle我们可以看是被Unity引擎压缩后的资源,其中就包含预制件、模型、图片、音乐等等,而LUA脚本跟C#一样,它用于控制游戏逻辑,只不过我们用LUA脚本来编写逻辑,就可以实现脚本的热更新.

LUA与C#在热更新上的区别,如果用一句话来说,目前在IOS上选择LUA来做热更相对成熟以及稳定.至于其它的一些说法,了解不了解对我们的学习没有太大的影响.以下是C#与LUA在热更新上的一些特定:

基于C#, 使用动态加载Assembly反射更新代码

这种方式在安卓上完全可行，对现有架构无需大的修改，一样使用C#和Unity3D的方式进行开发。但在iOS上受到限制，因此对于全平台首发的游戏，或者双平台都要上的游戏，已经慢慢的不使用这种方法进行热更新了。

基于Lua，将Lua代码视为资源，动态加载并运行

云风团队早期研究出的UniLua是基于C#编写的Lua虚拟机来运行， 而且只支持字节码解释，因此无法做动态功能，效率奇低。后期，ulua的出现，彻底将Lua作为比较正统的更新方式存在。ulua基于Tolua库进行封装，添加了一些便捷封装，代码打包和基本的框架。

原文链接:https://sanwen8.cn/p/283Bi8C.html

大家如果感兴趣可以去看看以上的那篇文章.

而在本节,我们要了解:

1.LuaForWindows_v5.1.4-35.exe 编程环境:
http://files.luaforge.net/releases/luaforwindows/luaforwindows
2.LuaStudio.exe 编码工具: http://www.luastudio.net/
3.(ulua)官网:http://www.ulua.org/index.html
4.TOLUA框架下载:https://github.com/jarjin/LuaFramework_UGUI
5.Lua官网: http://www.lua.org/

最后说下,本套课程的适应人群,如果不具备以下知识点,可能看后只知道一些理论,而无法自己进行实操,也无法在TOLUA这个框架上进行工作编码:

（1）至少会用Unity开发程序,基本的Unity的知识要懂,以及熟悉LUA语法.LUA语法可参

考:http://www.runoob.com/lua/lua-tutorial.html

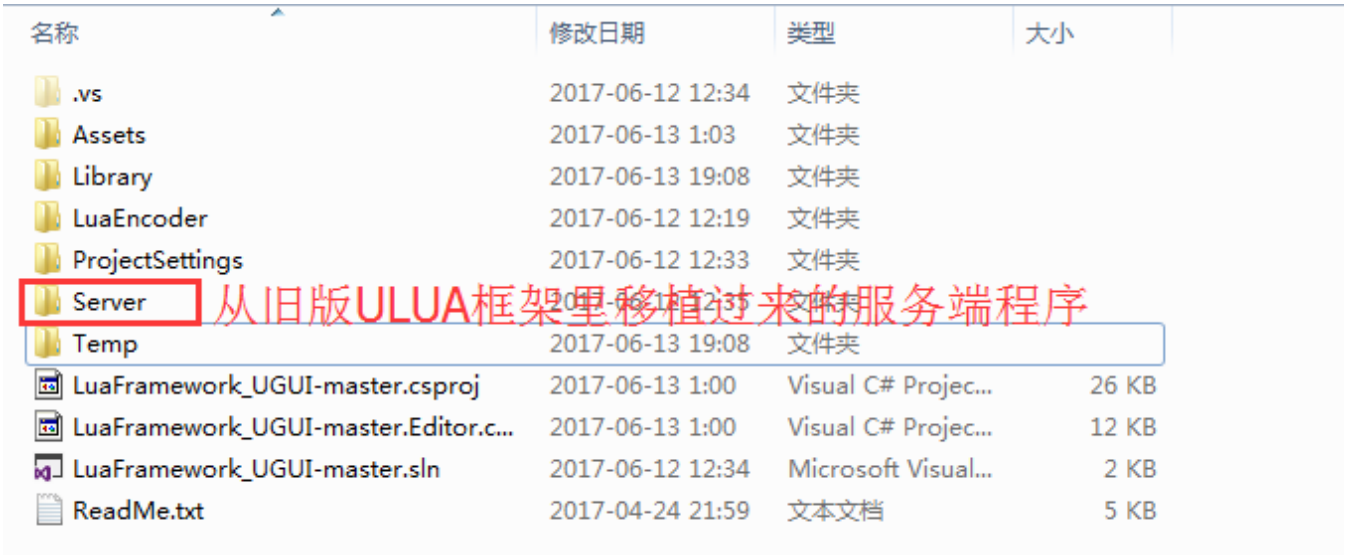
（2）知道怎么用Unity生成APK/IPA到移动设备上.

（3）知道 ULUA的c#与c、c++等原生插件通讯原理。可参考:http://www.cnblogs.com/warensoft/archive/2011/12/09/warenosoft3d.html

---

02.TOLUA_UGUI框架结构的简单介绍:

如下图所示,是框架压缩包解压后的目录,其中Server是我自己从旧版的ULUA框架里移植过来的.也可以用于TOLUA热更新项目的服务端,具体作用可参考课程ULUA的相关介绍,或者本节服务端配置的相关介绍.



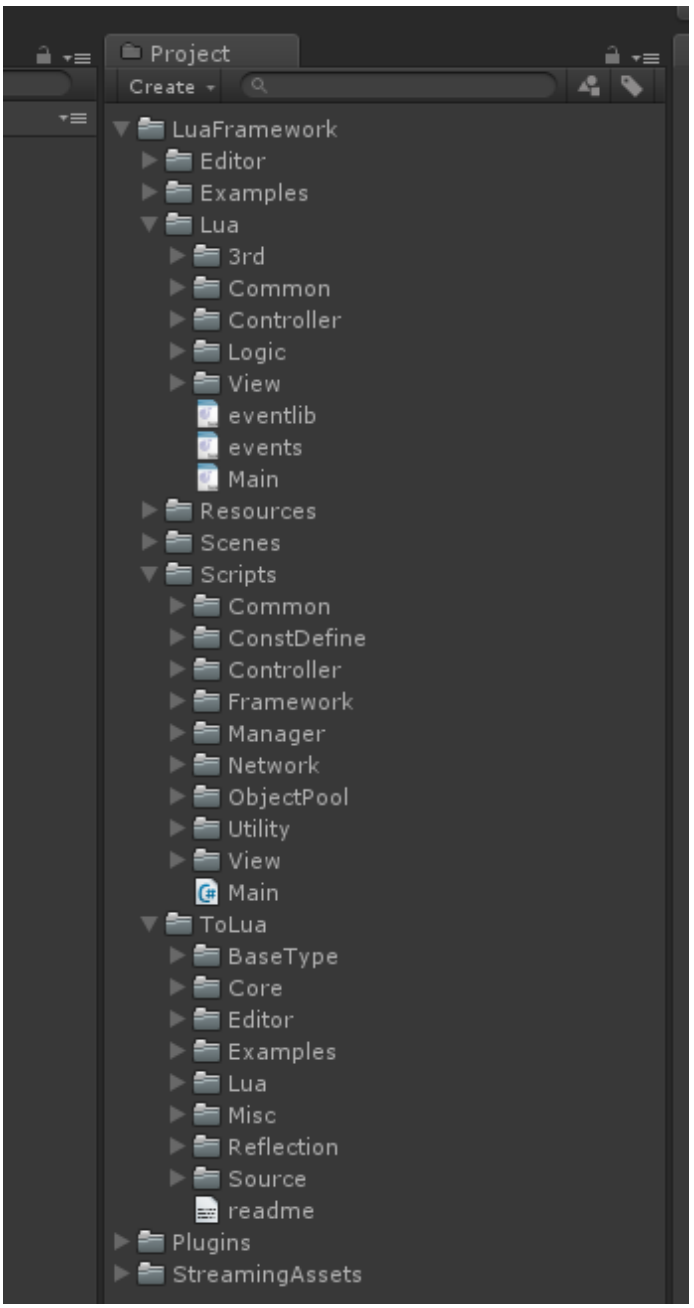
Unity工程的一般目录有:

目录、文件	说明
Assets	保存项目的图片、脚本、动画配置等等。一般是我们开发工程中的直接产物，需要提交到仓库管理起来。
ProjectSettings	所有通过Edit → Project Settings配置的信息都保存在这里。
obj、Temp	临时文件目录
Library	缓存文件。这个目录下的文件都会自动生成。
*-vs.csproj	Visual Studio IDE C#工程文件
*.csproj	MonoDevelop IDE C#工程文件
*-vs.unityproj	Visual Studio IDE JavaScript工程文件

*.unityproj	MonoDevelop IDE JavaScript工程文件
*-csharp.sln	Visual Studio 的解决方案
*.sln	MonoDevelop IDE 的解决方案

所以,主要介绍下框架的LuaEncoder文件,它是LUA的编码器,用于将LUA源码编译成二进制文件,以及各个平台解析lua代码的虚拟机.

接着是框架的目录的基本目录结构:



框架主要由三个主目录构成,分别是:

LuaFramework:存放框架相关的核心文件、示例工程等等

Plugins:存放各个平台所需要的类库

StreamingAssets:资源文件夹,通过菜单栏的LuaFramework下的Build XXX Resource而生成.该文件夹的所有资源需要放到资源服务器上,用于版本资源比较以及资源更新下载.

以下是对LuaFramework框架主目录里的文件夹进行介绍:

Editor:在该名称下的文件,都支持在编辑状态而非运行时候编译的脚本,其中CustomSettings用于生成wrap关联文件,Packager用于不同平台的打包.

Examples:存放自带例子所用到的素材,如预制件、图片以及执行欢迎界面的脚本.

Lua:框架自带的Lua源码目录，用户自定义的Lua脚本也就是放在这里面，最后打包的时候，打包脚本会将其按目录结构生成到StreaminAssets目录里面去，然后在将其上传到游戏的Web服务器上面，用于准备被每个游戏客户端下载更新他们本地的Lua脚本。达到热更目的。

在TOLUA里可以开启LuaBundle模式直接将lua文件打进Assetbundle.由AppConst脚本的LuaBundleMode布尔类型的变量决定.

---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protocal.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、NetworkManager网络管理器，如果你有新的管理器可以放到里面。

---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。

Resources:unity特殊文件夹,可以用于存放各种预加载的资源,生成不同平台的应用程序时,该文件夹下的所有资源一定会被打包.

Scenes:存放框架的示例main场景

Scripts:框架的C#脚本层,

---*Common*：框架的公用定义类。*LuaBehaviour*相当于*MonoBehaviour,*里面有各种特定的事件监听.*LuaLoader*与*lua*脚本文件的加载相关。

---*ConstDefine*：常量定义目录，*AppConst*（应用常量）*ManagerName*（管理器名称）*NotiConst*（通知常量，用于*mvc*消息通知）。

---*Controller*：控制器目录，*Command*常用的逻辑控制器。

---*Framework*：经过修改过的*PureMVC*的框架文件，*Core*核心类，*Interfaces*接口相关

---*Manager*：*Unity*提供基础功能的管理器类，音乐、面板、线程、资源等众多管理器。

---*Network*：网络的常用辅助类，*ByteBuffer*字节操作封装类，网络协议类，转换器类。

---*ObjectPool*：对想池，主要减少*GC*频繁调用，提高性能。

---*Utility*：常用工具类。

---*View*：*C#*用的*PureMVC*的视图层。

ToLua:ToLua的核心目录,以及一些基础用例.

*BaseType:*一些基础类型的绑定代码

*Core:*核心目录，所有*c#*与*lua*的交互都是通过它进行调度的。

Editor:（反射定义）生成*Wrap*文件列表的工具类目录。

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

*Examples:*框架自带的案例，提供了各种功能的演示。

*Lua:*框架所封装的一些*lua*工具。

*Misc:*存放一些杂项,杂项，目前有*LuaClient(*挂载这个脚本会启动*lua*框架的入口*) LuaCoroutine*（协程） *LuaLooper*（用于*tick*） *LuaResLoader*（用于加载*lua*文件）

*Reflection:*反射相关的脚本。

*Source:*用于存放动态生成的*LuaWrap*类，其实就是将*C#*注册到*lua*里,方便在*lua*里调用*C#*注册进来的类的一些成员*(*静态*).*

Pluagins目录下的文件:

Plugins：ulua底层库所在的目录，里面存放的是不同平台的底层库，之所以ulua效率高，就是它是纯c的lua虚拟机，而不是c#解释型的。

---*Andriod*：安卓*lua*虚拟机底层库，里面分为*armv7-a*与*Intel x86*平台。

---*iOS*：里面就是苹果*lua*虚拟机底层库。

---*ulua.bundle*：里面是*Mac*机器的底层库。

---*x86*：里面是*Win32/Linux32*位机器的*lua*虚拟机底层库。

---x86_64：里面是*Win64/Linux64*位机器的*lua*虚拟机底层库。

以上蓝色的名称，表示Unity里的特殊文件夹,有着特殊的意义.关于更多的特殊文件夹可以参考:http://wiki.unity3d.com/index.php/Special_Folder_Names_in_your_Assets_Folder

---

03.菜单栏及课程学习目标:

如下图所示所示,框架在菜单栏做了2个扩展,一个是LuaFramework菜单,另一个是Lua,前者主要用于生成不同平台下的资源,后者主要用于清除和生成wrap文件相关.

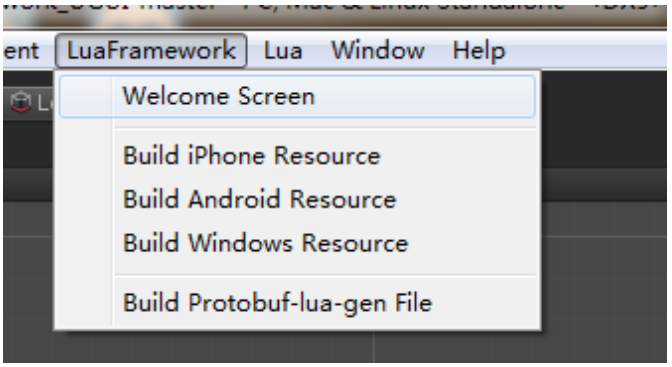
Welcome Screen--执行的代码在WelcomeScreen脚本里,主要是新手的一些提示.

Build iPhone Resource--生成IOS平台的Assetbundle

Build Android Resource--生成安卓平台的Assetbundle

Build Windows Resource--生成Windows平台的Assetbundle

Build Protobuf-lua-gen File--生成Protobuf-lua文件



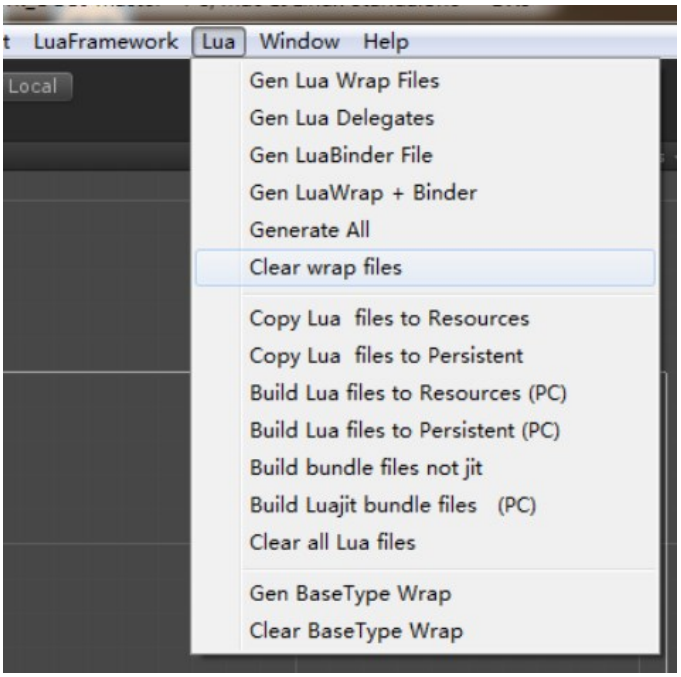
在下图lua菜单里的选项所执行的代码在ToLuaMenu这个脚本里.

我们常用的有:

Generate All 生成所有相关wrap文件

Clear wrap files 清除生成的wrap文件

其它的大家可以根据名称以及ToLuaMenu脚本查看相关的代码.



本套课程的学习目标:

通过这个项目我们要学会,代码热更新的原理,以及UI如何做热更新,三维模型如何做热更新.以及如何将资源上传到阿里云服务器,架设web站点进行公网测试.

本节任务:

在框架的示例工程的Min组件里的AppFacade.Instance.StartUp();这个语句加一个断点进行工程的调试,查看整个代码执行的流程,重点需要关注GameManager.cs这个脚本里的初始化函数OnInitialize,同时要查看Game.lua实现OnInitialize所调用的方法.另外PanelManager这个脚本也可以查看下,它是用于UI面板的创建和管理.

04.怎么通过lua代码向控制台输出消息以及代码更新原理:

任务:

01.实操:通过lua代码,在控制台输出"Hello码锋学院2017"

02.解决疑问:如何进行代码的热更新?

输出"Hello码锋学院2017"--**具体操作:**

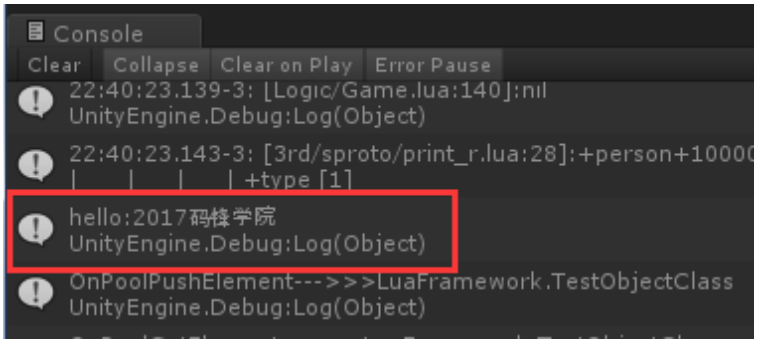
01.新创建一个工程,创建一个Canvas,并且将标签改为GuiCamera(或者将PanelManager.cs里的第14行,GuiCamera改为Canvas自定义的标签,必须是唯一的),并且创建一个空物体,名称为:GameManager,挂载Main脚本.

02.打开Game.lua脚本,在Game.OnInitOK()方法里,加上一行代码:

```
1  LuaFramework.Util.Log("hello:2017码锋学院")
```

03.点击菜单栏,LuaFramework-Build Windows Resource(此操作会生成window平台的资源,包括lua代码与assetbundle).

04.运行项目,则可以看到控制台的输出,如下图:



注意的地方:

01.AppConst.cs里可以把LuaBundleMode设置为false,直接读取lua脚本的代码,可方便我们开发时候调试,不用每次进行打包-Build xxx Resource.

02.我们每次修改AppConst.cs后,要进行重新第一次运行工程前,需要点击Lua菜单-Clear wrap files清除相关的包装文件,等待执行完毕之后,点击Generate All进行重新生成.再进行buidl对应平台的resources,否则运行项目有可能出现报错的情况.

如何进行代码的热更新?

01.在Build xxx Resource资源的时候,其实就是将一些资源打包成assetbundle文件,存放在StreamingAssets文件夹下,这个文件夹需要用于配置资源站点.

02.项目每次运行的时候,会与该站点的文件做比较,然后下载以及加载这些文件,而lua脚本也是一样.

03.要进行代码的热更,则需要修改lua脚本里的代码,然后重新build,将新的资源上传到服务器指定的文件夹下.

04.下次项目启动的时候,游戏检测到资源有变更,则会下载这些新的资源(其中就包括lua脚本),执行新的lua代码,就实现了代码的热更新.

05.测试:AppConst.cs脚本里的LuaBundleMode保持为true,然后把上面添加的输出语句进行修改,输出其它信息,不重新打包的情况下去运行项目,发现并不会执行新的代码.

06.具体详细的测试,可以在后面学习了配置站点资源后,进行测试.

本节任务:

01.查看Packager.cs脚本,重点关注HandleExampleBundle方法是如何对文件夹里的资源进行打包的.

02.查看LuaHelper.cs脚本,重点关注GetResManager方法,以及该方法返回的对象ResourceManager,是如何调用LoadPrefab方法进行资源的加载.

05.如何通过lua代码加载3D模型以及模型更新原理?

任务:

01.实操:导入资源包NPC,通过lua代码加载3D模型,Boy_prefab到场景中.

02.问题:如何进行3d模型的热更.

实操:

01.新创建一个工程,创建一个Canvas,并且将标签改为GuiCamera(或者将PanelManager.cs里的第14行,GuiCamera改为Canvas自定义的标签,必须是唯一的),并且创建一个空物体,名称为:GameManager,挂载Main脚本.并且检查AppConst.cs脚本的LuaBundleMode = true.

02.导入资源包NPC.unitypackage.

03.在Packager.cs脚本里的HandleExampleBundle方法,添加以下代码,在打包资源的时候就可以将导入的资源包NPC/prefab文件夹下的prefab打包为assetbundle资源.

```
1  //将Assets/NPC/prefab这个文件夹下的prefab打包为assetbundle,名称为npc.unity3d
2  /AppConst.ExtName=.unity3d
3  AddBuildMap("npc" + AppConst.ExtName, "*.prefab", "Assets/NPC/prefab");
```

04.在Game.lua脚本里的Game.OnInitOK()方法添加以下代码,进行加载资源:

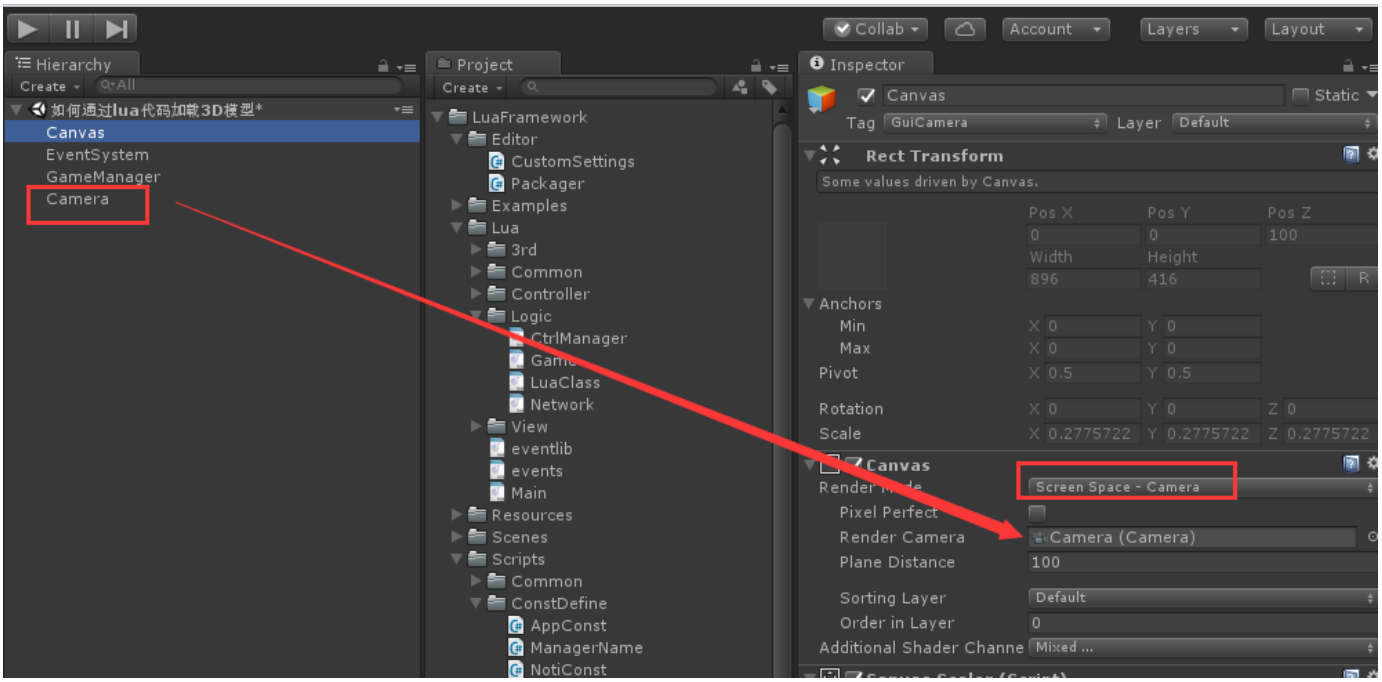
```
1  --资源加载的过程
2      LuaHelper = LuaFramework.LuaHelper;
3      resMgr = LuaHelper.GetResManager();
4      --第一个参数为assetbundle的名称,第二个参数为预制件名称,第三个参数是加载完成的回
   调
```

```
5       resMgr:LoadPrefab('npc', { 'Boy_prefab' }, OnLoadFinish);
```

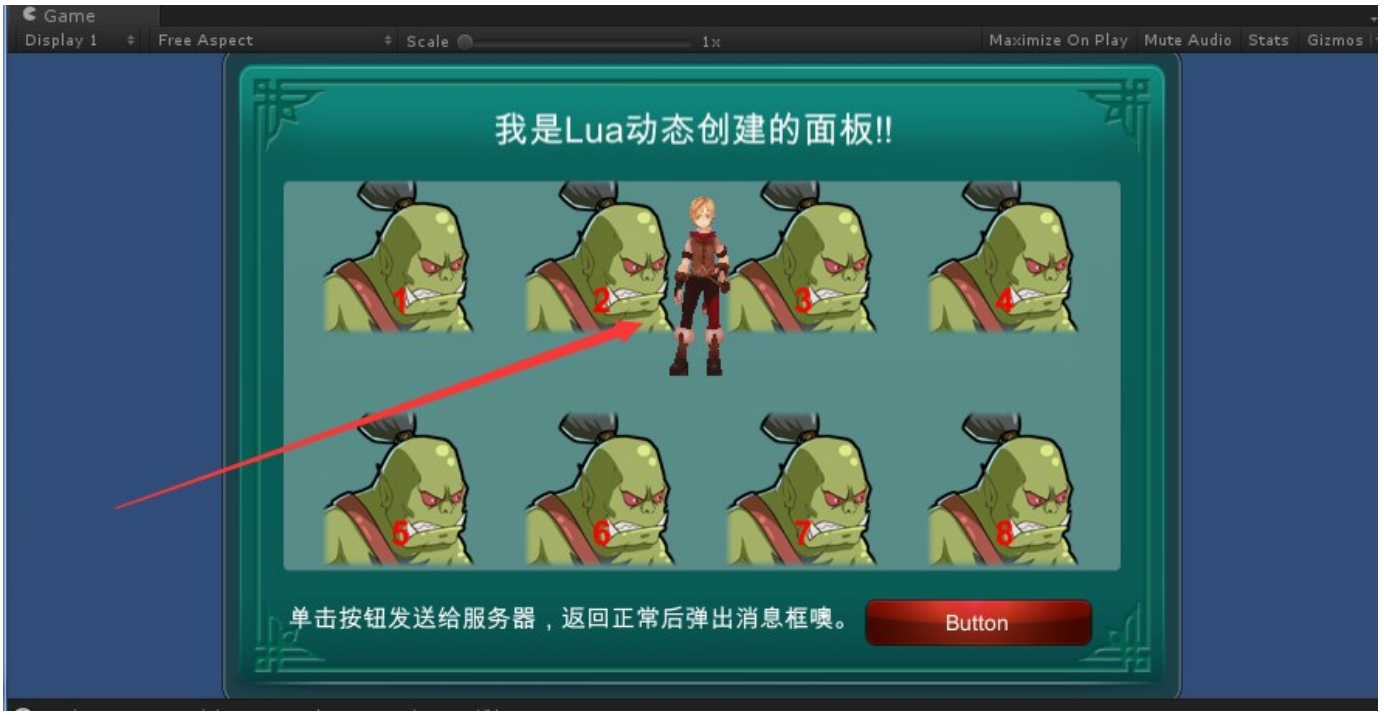
05.实现上一个步骤中调用的回调,也就是在Game.lua脚本里添加OnLoadFinish方法,进行实例化对象到场景中,代码如下:

```
1  function OnLoadFinish(objs)
2      --克隆加载的prefab到场景里
3      local go = UnityEngine.GameObject.Instantiate(objs[0]);
4      LuaFramework.Util.Log("npc加载完成,如果要在加载完成执行相关逻辑,可在此方法添
   加")
5  end
```

06.修改Canvas对象,将Canvas组件上的Render Mode(渲染模式)修改为Screen Space - Camera.并在场景创建一个Camera,将Tranform组件进行Reset,然后指定给与Canvas组件.并且修改NPC里的预制件Boy_prefab与girl_prefab,Z轴为3,避免加载的时候无法渲染到相机范围内.



07.点击Lua菜单-Clear wrap files清除相关的包装文件,等待执行完毕之后,点击Generate All进行重新生成.再进行buidl windows resources,然后运行工程,效果如下图:



08.可以查看StreamingAssets文件夹,里面就有npc.unity3d名称的assetbundle文件,而在Hierachy视图,看到Boy_prefab(Clone)对象,这个就是通过lua代码克隆出来的.

09.如果我们在这里不想加载框架原来的UI面板,可以把Game.lua脚本里的第53行注释掉.重新执行07步骤就可以了.

问题解决:

模型的更新与代码更新原理是一样的,代码修改的是lua脚本,而模型则需要修改对应的预制件,assetbundle名称与预制件名称保持一致的情况下,去修改预制件的网格模型、贴图等,然后进行重新打包,将新的资源更换服务器站点的资源,重启客户端就会下载新的资源文件(不管是lua脚本还是3d模型,准确的讲应该是预制件,以及下一节课讲的UI也是同样的原理.)

具体操作过程,可参考后面资源服务器站点配置的课程.

本节任务:

01.了解Game.InitViewPanels方法,是如何引入View下的xxxPanel脚本.重点关注下方法中的PanelNames,它是define脚本中定义的对象,存储所有面板的名称.

02.同时关注下define脚本中的CtrlNames,该对象存储了各个控制器(lua脚本)的名称.

03.关注下CtrlManager.lua脚本里的CtrlManager.Init()方法.

04.了解下define脚本的panelMgr对象,它可以调用CreatePanel进行创建面板.

05.了解下require关键字的作用.

06.查看下PromptCtrl以及PromptPanel的2个lua脚本,分别实现了什么功能.

07.了解下LuaBehaviour中的AddClick方法,是如何进行按钮点击监听的.

06.如何通过lua加载UI面板以及按钮点击事件的监听:

说明:

01.在该框架中,UI通过面板进行管理,一个面板下可以有多个不同的UI元素.每一个面板由2个lua脚本进行管理,一个用于存储视图元素,以xxxPanel.lua0进行命名,存放在Lua/View目录下;另一个则用于流程控制或者其它的逻辑事件,如按钮点击的监听,以XXXCtrl.lua命名,存放在Lua/Controller文件夹下.

02.每一个UI面板也需要做成预制件,然后打包为assetbundle文件,如果要更新,则更新预制件,重新打包assetbundle文件即可,之后上传到资源服务器上即可.

任务:

01.实操:通过lua代码创建一个UI面板,其中包括UI资源中的一张图片bg01.以及面板下有一个按钮,每次点击会将Image组件(bg01)的状态设置为相反的.

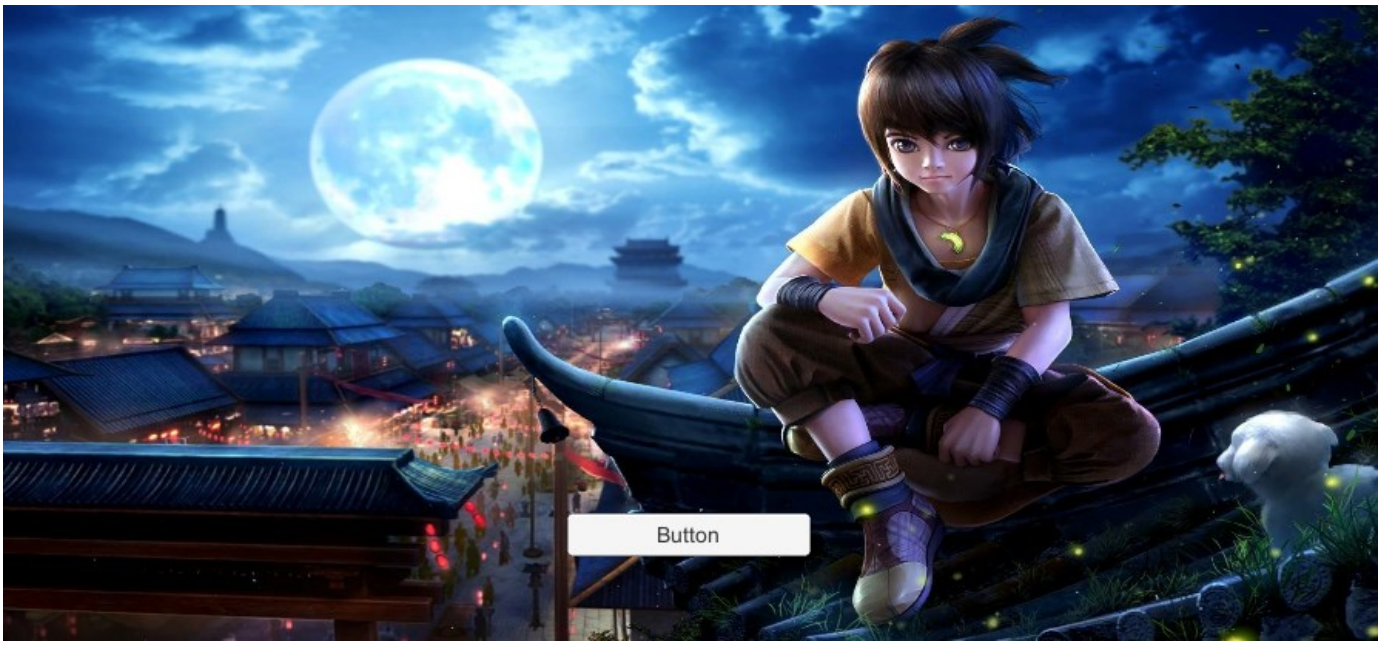
实操:

01.新创建一个工程,创建一个Canvas,并且将标签改为GuiCamera(或者将PanelManager.cs里的第14行,GuiCamera改为Canvas自定义的标签,必须是唯一的),并且创建一个空物体,名称为:GameManager,挂载Main脚本.并且检查AppConst.cs脚本的LuaBundleMode = true.

02.导入资源包UI.unitypackage,处理UI的打包,在Packager.cs脚本里的HandleExampleBundle方法,添加如下代码:

```
1   //将Assets/UI/prefab这个文件夹下的prefab打包为assetbundle,名称为bg.unity3d
2  AddBuildMap("bg" + AppConst.ExtName, "*.prefab", "Assets/UI/prefab");
```

03.将UI/prefab目录里的BGPanel拖放到场景的Canvas对象下,这是我预先做好的一个预制件,这里我们要检查图片是否能正常显示,如下所示:



04.创建BGPanel预制件对应的2个lua脚本,一个为BGPanel.lua,存放在Lua/View目录下,该脚本缓存BGPanel下的所有对象,这里包括Image与button2个元素.代码如下:

```lua
local transform;
local gameObject;

--定义一个table,table是lua中的一种数据结构,可以存储多种不同的对象以及方法
--可以将它看似C#中的类
BGPanel = {};


--this指向了table对象
local this = BGPanel;

--Panel的Awake方法,会在Ctrl里加载
function BGPanel.Awake(obj)

    gameObject = obj;
    transform = obj.transform;

    this.InitPanel();
end


--初始化面板--
function BGPanel.InitPanel()
    --btn按钮初始化
    this.btnObj = transform:FindChild("btn").gameObject;
```

```
26      --Image初始化
27      this.image=transform:GetComponent('Image');
28  end
```

05.另一个lua脚本为BGCtrl.lua,存放在Lua/Controller下.代码如下,以下这个脚本的注释非常重要,它可以帮助你理解具体的面板创建流程:

```
1   require "Common/define"
2
3   BGCtrl = {};
4   local this = BGCtrl;
5   local transform;
6   local gameObject;
7   local lua;
8
9   --你可以把它理解为构造函数,但其实与C#的构造函数还是不同的,它需要被调用.
10  --具体调用可以参考CtrlManager.Init()方法.
11  function BGCtrl.New()
12      --这个引用其实是被CtrlManager.Init()中的某个对象指向了.
13      return this;
14  end
15
16  function BGCtrl.Awake()
17      --创建面板,当它创建的时候,会自动为面板添加LuaBehaviour组件
18      --LuaBehaviour又会根据当前面板名称,调用xxxPanel.lua这个脚本的Awake和Start方
    法.
19      --具体可以查看LuaBehaviour.cs这个脚本
20      --为了它能够调用(BG)xxxPanel方法,所以这里的名称一定要与(BG)xxx一致.
21      --同时这个参数也是用于加载ab(assetbundle))文件的,具体可以参考PanelManager.cs
    脚本的第27行
22      --它先转化为小写,再+上AppConst.ExtName里的设置,就等于ab的名称
23      panelMgr:CreatePanel('BG', this.OnCreate);
24      --所以,这里创建后的面板名称为:BGPanel
25      --所以,这里创建后加载的ab名称为bg.unity3d
26  end
27
28  function BGCtrl.OnCreate(obj)
29      gameObject = obj;
30      transform = obj.transform;
31
```

```lua
    --添加点击事件
    lua = transform:GetComponent('LuaBehaviour');
    lua:AddClick(BGPanel.btnObj,this.OnClick);


end

b=true;

function BGCtrl.OnClick(go)

    print("点击了按钮,设置精灵组件的状态");

    --每次点击都执行以下这段代码,表示:
    --先判断b的值,然后设置为相反,把Image组件的状态设置=b
    if b==true then
        b=false;
        BGPanel.image.enabled=b;
    else
        b=true;
        BGPanel.image.enabled=b;
    end


end
```

06.在define.lua脚本的CtrlNames与PanelNames中新增添加的控制器与视图,具体代码如下,其中PanelNames会在Game.lua脚本里的Game.InitViewPanels方法使用到.而CtrlNames则在CtrlManager.lua中的CtrlManager.Init()使用到.

```lua
CtrlNames = {
    Prompt = "PromptCtrl",
    Message = "MessageCtrl",
    --BG是新增的,有多少个控制器就添加多少个
    BG="BGCtrl",
}

PanelNames = {
    "PromptPanel",
    "MessagePanel",
```

```
    --BGPanel是新增的,有多少个新增的Panel就新增多少个
    "BGPanel",
}
```

07.在CtrlManager中,通过require引入Controller/BGCtrl,同时修改CtrlManager.Init()方法,添加新增的BG对象.代码如下:

```
require "Controller/BGCtrl"

function CtrlManager.Init()
    logWarn("CtrlManager.Init----->>>");
    ctrlList[CtrlNames.Prompt] = PromptCtrl.New();
    ctrlList[CtrlNames.Message] = MessageCtrl.New();
    --新加的控制器对象,调用了BGCtrl.New()
    --注意BGCtrl.New()返回的是this,this又指向了BGCtrl里的BGCtrl这个Tabel
    ctrlList[CtrlNames.BG] = BGCtrl.New();
    return this;
end
```

08.在Game.lua中,引入BGCtrl,代码如下:

```
require "Controller/BGCtrl"
```

09.在Game.lua中,修改Game.OnInitOK()方法,主要为第53行,CtrlNames换为BG.

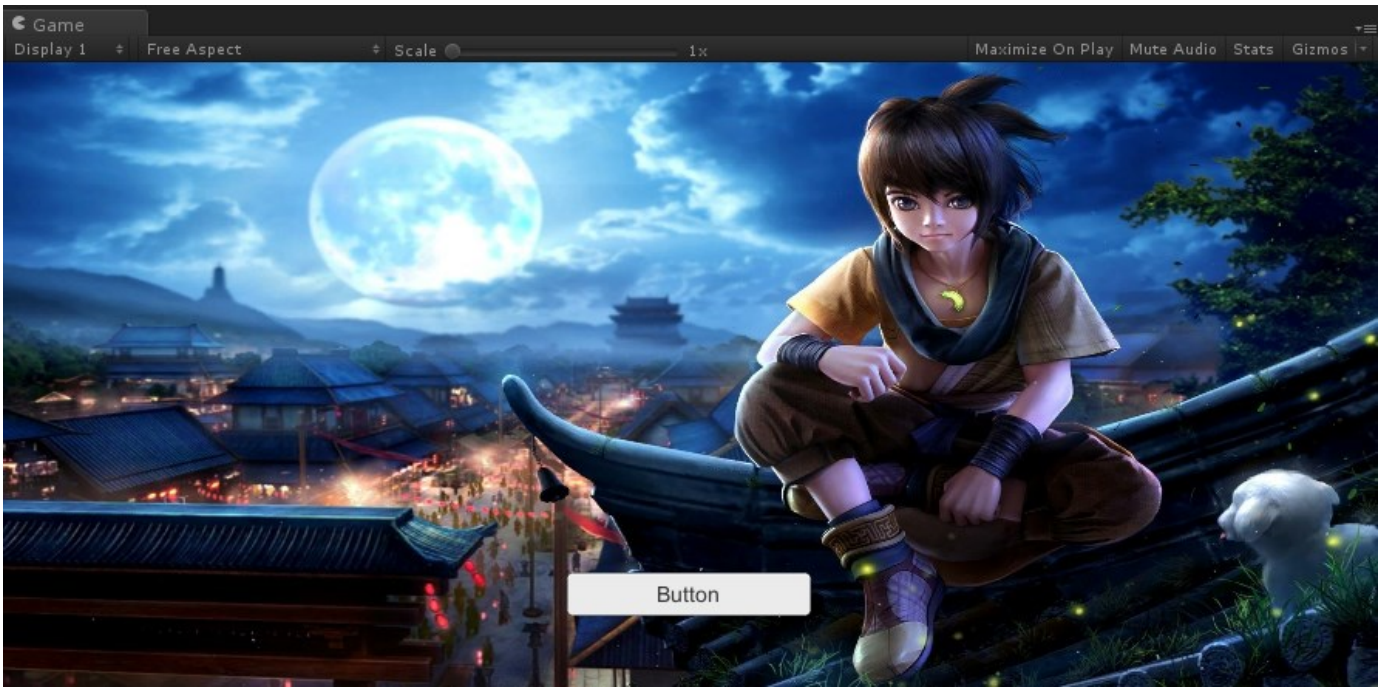
```
local ctrl = CtrlManager.GetCtrl(CtrlNames.BG);
    if ctrl ~= nil and AppConst.ExampleMode == 1 then
        ctrl:Awake();
    end
```

10.(可选操作)在Game.lua中,一些test相关的代码,以及我们之前课程写的代码,都可以注释掉,或者删除掉.跟我们UI创建并没有多大的关系.仅保留以下代码即可:

```
function Game.OnInitOK()
    AppConst.SocketPort = 2012;
    AppConst.SocketAddress = "127.0.0.1";
    networkMgr:SendConnect();
```

```
5
6       --注册LuaView--
7       this.InitViewPanels();
8
9       CtrlManager.Init();
10      local ctrl = CtrlManager.GetCtrl(CtrlNames.BG);
11      if ctrl ~= nil and AppConst.ExampleMode == 1 then
12          ctrl:Awake();
13      end
14
15  end
```

11.点击Lua菜单-Clear wrap files清除相关的包装文件,等待执行完毕之后,点击Generate All进行重新生成.再进行buidl windows resources,然后运行工程,效果如下图:



12.点击按钮即可将背景图片进行隐藏/显示.

本节任务:(重要)

01.总结UI的创建以及加载流程.特别需要注意的是,define.lua、game.lua、ctrlManager.lua这三个脚本里需要新增以及修改的地方.

02.一定要去看看PanelManager.cs这个脚本调用创建面板的方法(CreatePanel),涉及到面板名称的定义,加载ab(assetbundle)文件,以及为面板添加LuaBehaviour组件.

03.同时也要注意在LuaBehaviour组件(脚本)中,又会执行Awake方法调用View/XXXPanel.lua脚本的Awake方法.其最终意义就是要缓存视图下的所有对象.比如按钮.

04.在控制器脚本中,使用LuaBehaviour调用AddClick方法进行按钮点击监听的时候,又会用到XXXPanel中的UI元素.

05.虽然有点复杂的样子,但也只能熟悉多看几次啦,否则就自己重写框架里创建、管理面板的流程.

07.如何通过阿里云配置资源服务器?

说明:

阿里云服务器,我们可以简单的理解为一台远程的主机(电脑),配置有一个公网IP,可提供给网络上的其它用户进行访问,通过端口就可以访问该主机开放下载的一些文件资源.而这个步骤可以通过IIS管理器配置站点来完成,站点指向要被下载的资源目录.

任务:

01.通过阿里云配置资源站点;

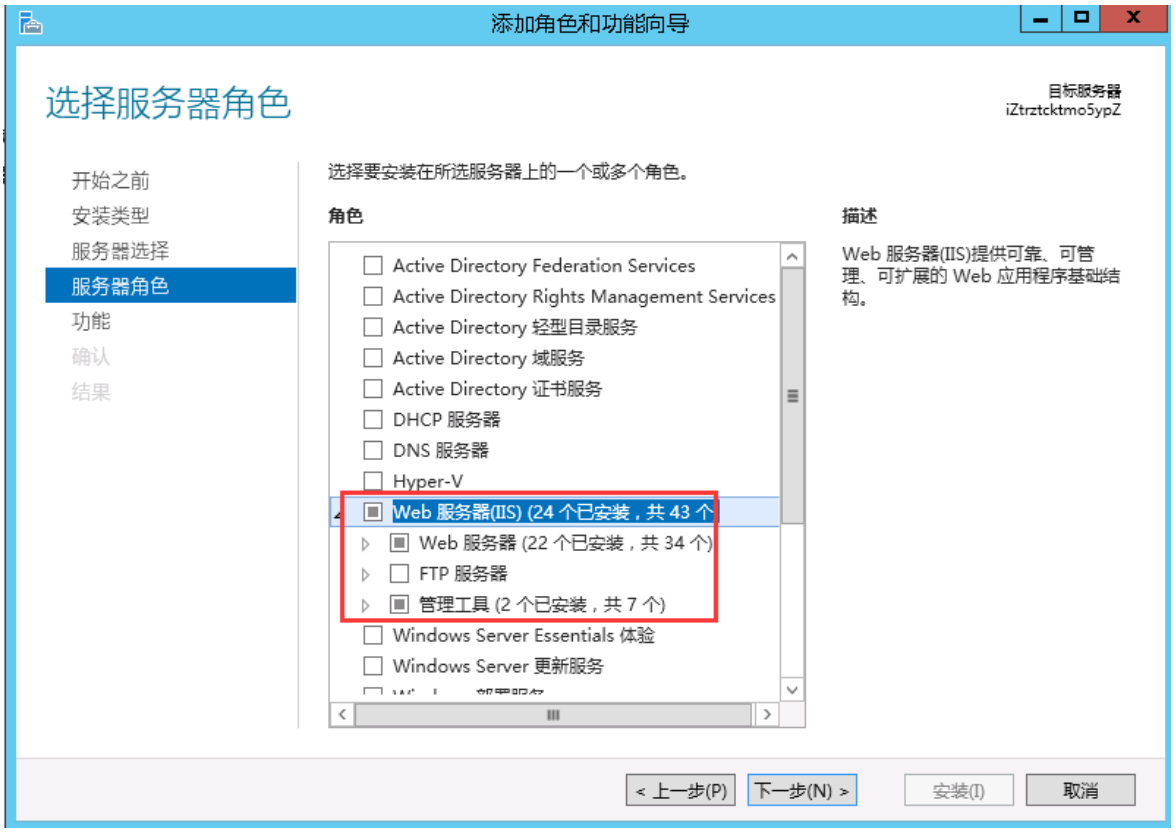
02.工程进行资源热更新测试;

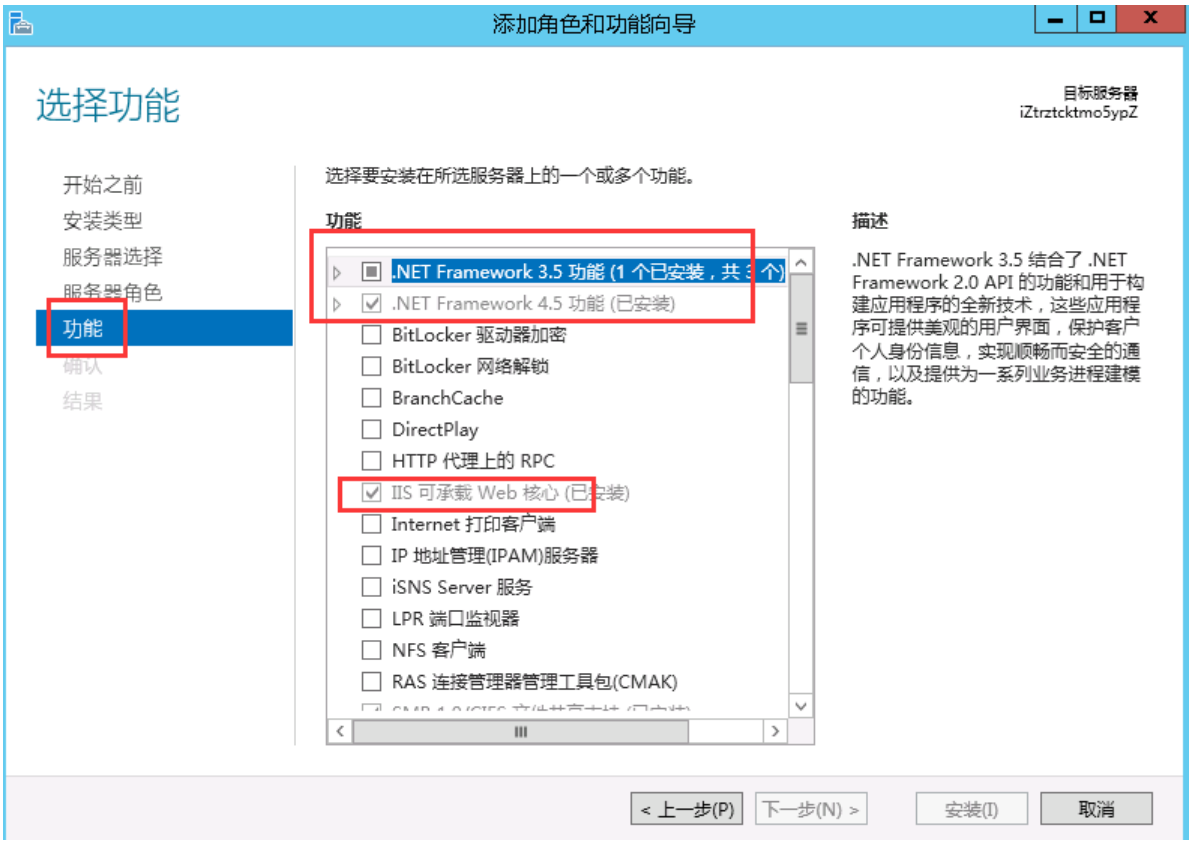
实操:

01.在阿里云购买ECS服务器,并且选择Windows Server镜像(系统):https://wanwang.aliyun.com/ 之后进入阿里云控制台,启动远程的主机以及查看对应的公网IP.

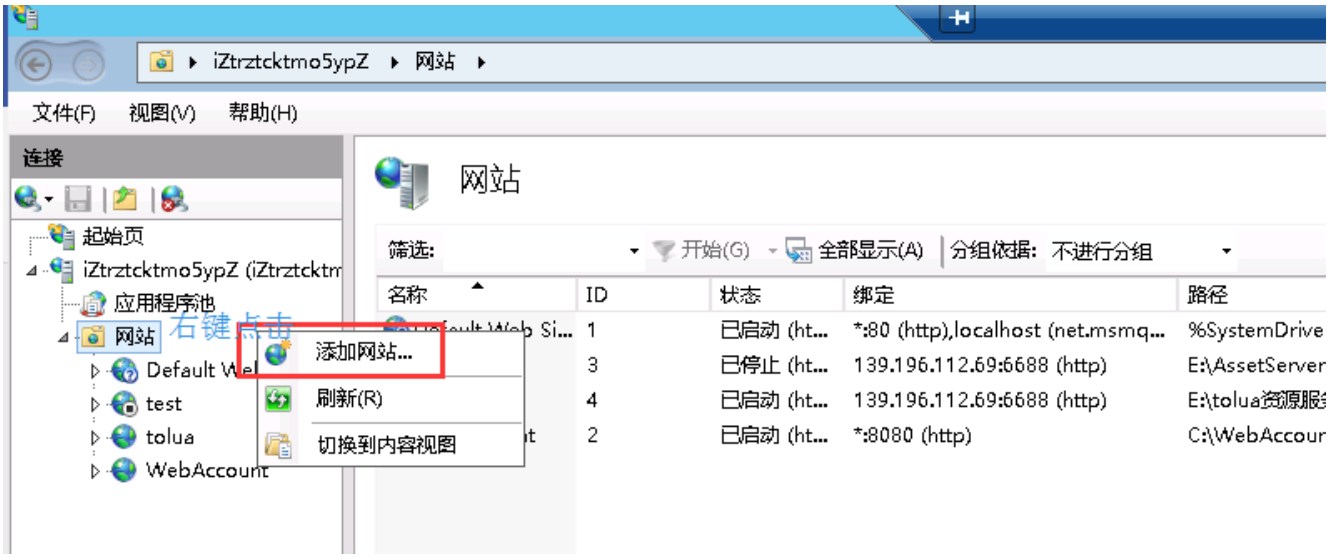
02.我们可以通过本地电脑自带的[远程桌面连接]服务进行连接阿里云购买的服务器.也可以在控制台通过网页的形式进行远程连接.

03.连接进入系统后,需要安装.net,百度搜索下进行安装即可.然后在仪表盘中添加角色和功能,根据提示进行添加IIS管理器.





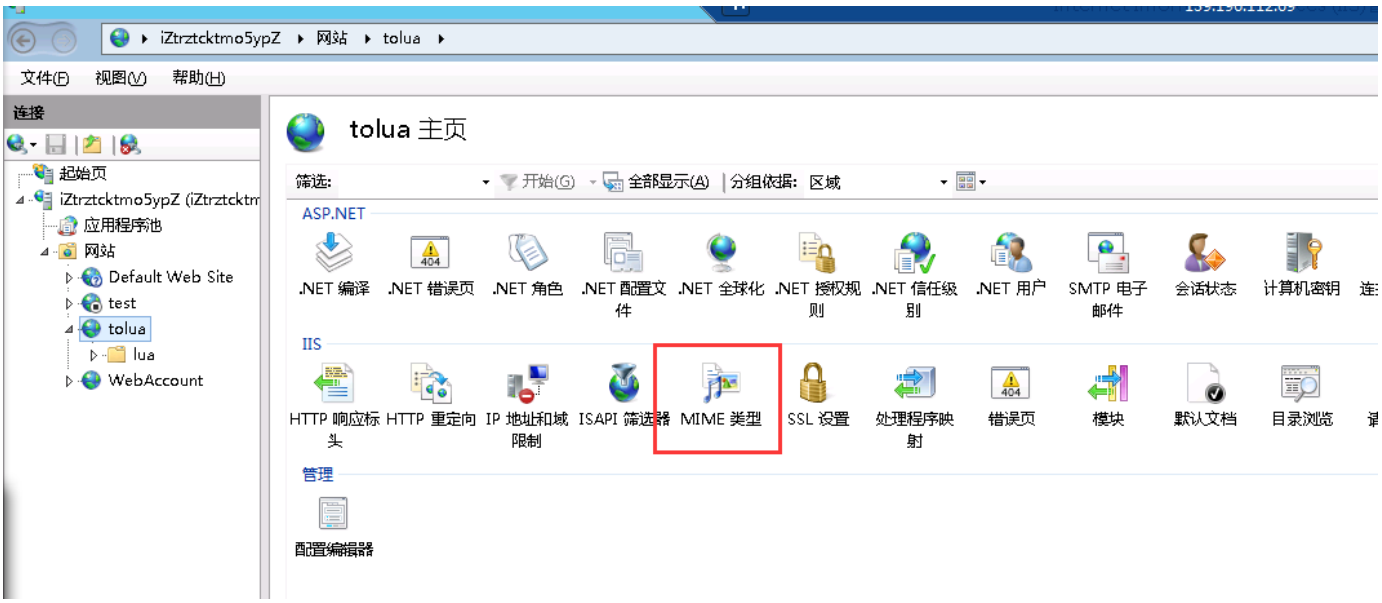
04.添加完毕之后,则可以通过IIS管理器进行创建网站,创建的时候需要添加IP和端口,IP为公网IP,端口设置为与我们客户端配置的端口一样:6688.



05.需要将Unity工程里打包好的StreamingAssets文件夹上传到阿里云主机上的硬盘,上传的方法超多,可以百度下.我习惯直接在远程连接里配置共享本地硬盘的数据,但此方法有一定的风险,因为当你服务器被黑客入侵之后,你电脑上硬盘的数据就显得很脆弱了.所以如果选择该方法,建议只共享需要的数据盘即可,避免造成巨大损失.上传之后指定物理路径为StreamingAssets目录即可.就可以点击确定,此时站点就创建成功了,但还需要进一步配置权限.

06.在站点主页,双击选择IIS-MIME类型,然后进行操作添加扩展名与类型.依次添加以下扩展名,如存在则忽略:

```
添加：扩展名.assetbundle  类型application/octet-stream (类型为字节流)
添加：扩展名.txt  类型application/octet-stream
添加：扩展名.manifest  类型application/octet-stream
添加：扩展名.（空格）    类型application/octet-stream
添加：扩展名.lua  类型application/octet-stream
添加：扩展名.*    类型application/octet-stream
```



07.回到unity工程,打开AppConst.cs脚本,修改DebugMode=true,UpdateMode = true,表示开启更新模式并且进行本地调试,再将WebUrl修改为服务器对应的公网IP,我的如下:

```
public const string WebUrl = "http://139.196.112.69:6688/";       //测试更新地址
```

08.启动上一节创建的工程,加载出来的资源是一样的.

09.此时我们可以修改UI/prefab下的BGPanel预制件,将Image图片更换为bg02或者bg03,然后进行重新打包.

10.此时我们要注意,如果没有开启更新模式,肯定是会加载本地新打包的资源,但是如果开启了更新模式,则无论本地资源如何变化,还是以服务器的资源为主.不信的话,我们可以先不上传资源到服务器,直接打包后运行项目,则会发现本地客户端从服务器去下载之前上传的资源,并且加载之前的资源.此时,我们要的答案已经有了,是否再将资源上传到服务器测试已经不重要啦~~

11.不过我们还是可以试一试,注意的是,上传新的资源到服务器之后,站点的物理路径要重新指定或者配置下,每次对IIS物理路径进行操作的时候,都会导致IIS站点异常.

12.我们不仅可以修改UI预制件,也可以修改LUA代码,比如注释掉按钮的点击功能等,然后再重新打包上传到服务器,测试是否会进行代码的更新.答案是肯定的,所以我在这里就不进行演示啦.

其实....下一节将演示真机测试,还是可以看到这个过程的.

08.通过真机演示热更新过程:

说明:

01.不同的平台,对应的Assetbundle文件是不同的.需要重新打包并且重新配置站点.

02.在真机测试需要将AppConst.cs的DebugMode设置为False.

任务:

01.发布到手机上进行测试.

02.更换站点资源,重启APP演示热更新过程.

03.测试场景为第六课里创建的场景.

操作:

01.修改AppConst.cs的DebugMode=False

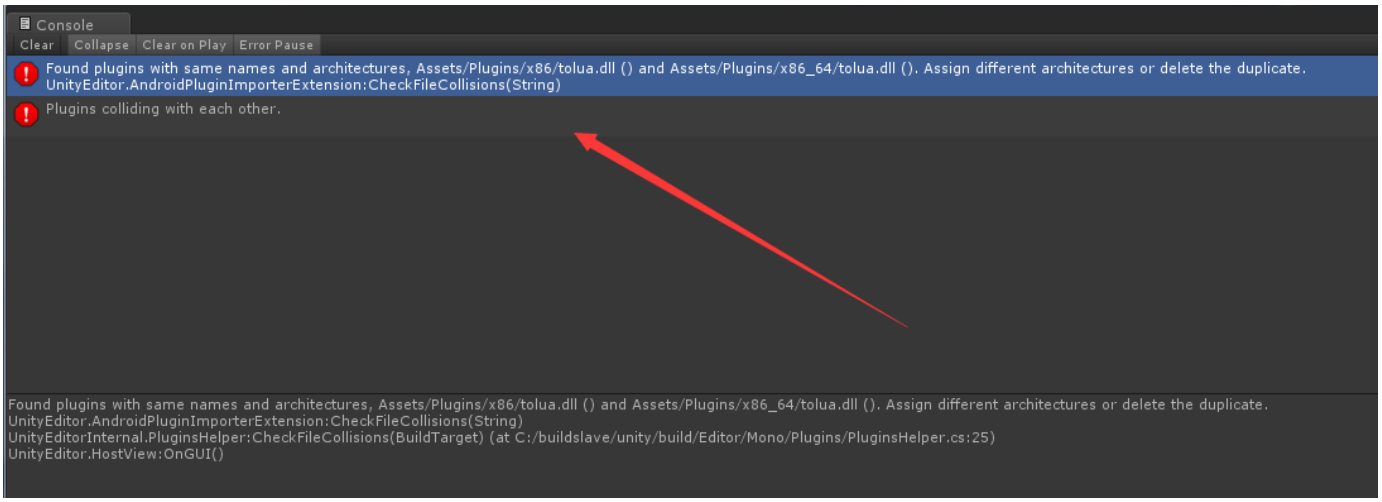
02.删除StreamingAssets文件夹,并按快捷键Ctrl+B打开发布设置,切换为安卓平台.

03.点击Lua菜单-Clear wrap files清除相关的包装文件,等待执行完毕之后,点击Generate All进行重新生成.再进行buidl android resources.

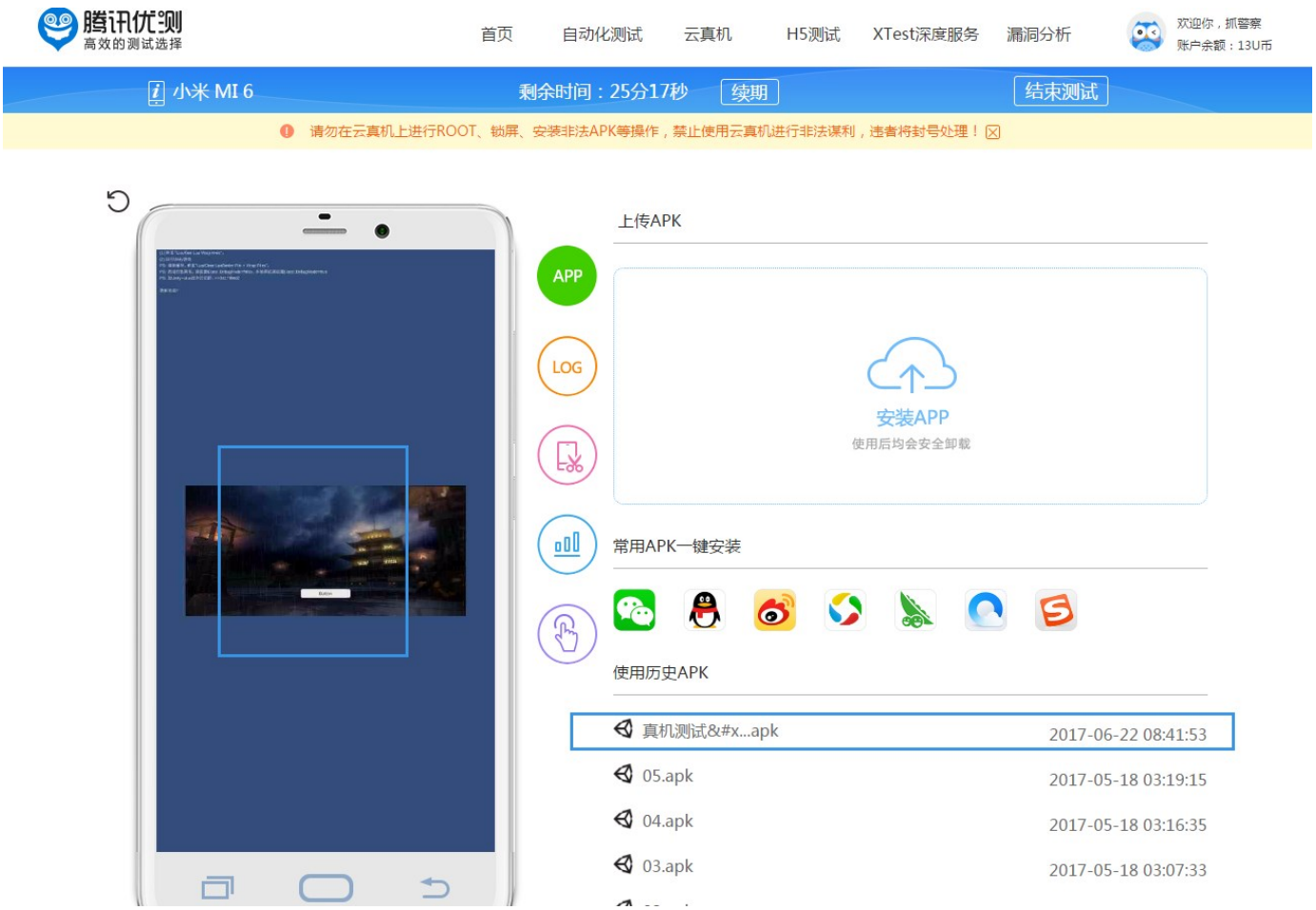
04.将新的StreamingAssets文件夹替换阿里云服务器上对应的文件夹,重新配置MIME类型里的文件扩展名.

05.打包安卓平台可执行的APK包,记得添加场景为06课创建的场景,在这个工程中,我给他起了个好名字:"如何通过lua加载UI面板以及按钮点击事件的监听"

06.打包时候报错是正常的,如下,这种情况的解决方案是:如果是32位Unity编辑器，删掉Plugins下的x86_64目录，如果是64位编辑，删除Plugins下的x86目录即可.



07.打开腾讯优测:http://utest.qq.com/ 选择机型,将APK包上传安装.最终运行结果如下,因为在打包的时候,BGPanel指定的是bg02,所以就是如图的样子了,点击按钮可以进行设置图片的显示和隐藏.

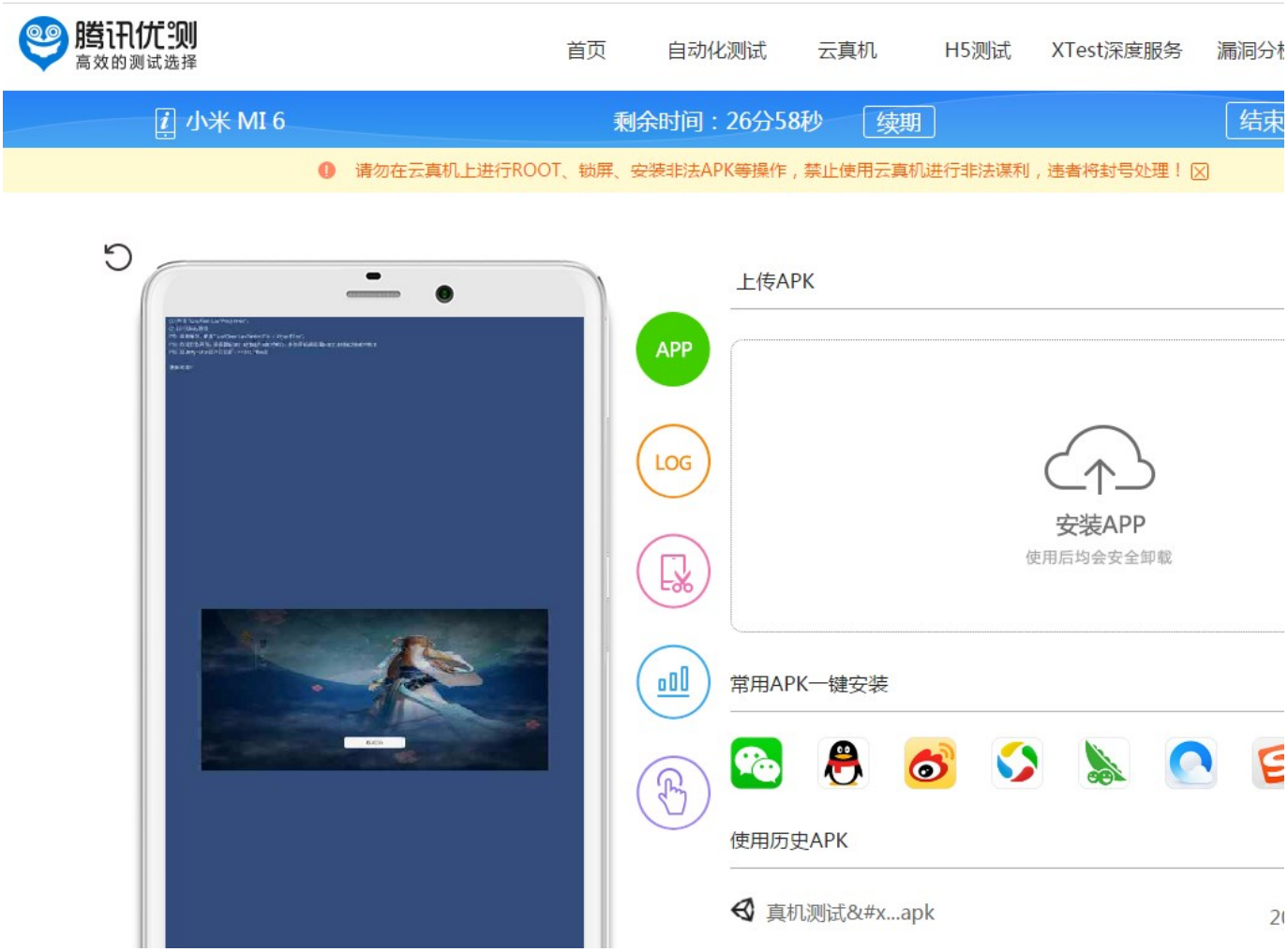


08.回到unity工程,将BGPanel图片更换为bg03.打开BGCtrl.lua,注释掉第34行:

```
1  --lua:AddClick(BGPanel.btn,this.OnClick);
```

09.回到unity工程,重新打包,将新的资源替换服务器旧的资源.重新启动手机的客户端程序进行测试查看.

10.测试结果如下,bg图片替换了,而按钮的点击功能也失效了:



11.在无需重新更换APK包的情况下,我们就实现了资源以及代码的热更新.

补充:

在unity工程中,为Camera添加AppView组件,就可以看到一些界面提示,如资源实时下载信息,解包过程等等.

9.lua如何调用C#代码?

任务:

01.创建测试脚本,里面提供2个变量,一个静态、一个非静态的.同时也提供2个方法,一个静态的,一个非静态的.之后在Main.lua里去调用这2个变量和2个方法.

lua调C#:

这里我们要考虑2个情景:

情景1,C#脚本继承 MonoBehaviour,那就需要先将该脚本作为组件,挂载到场景对象上,才可调用一些非静态的变量和方法,测试代码如下:

```
using UnityEngine;

public class TestCall:MonoBehaviour  {
    public static string str01 = "这是测试脚本里的第一个变量:静态的,--继承Mono--
";
    public  string str02 = "这是测试脚本里的第二个变量:非静态的,--继承Mono--";


    public static void AddCom(GameObject go) {
        go.AddComponent<TestCall>();
    }

    public static void Function01() {

        Debug.Log("调用了第一个方法:静态的,--继承Mono--");
    }

    public  void Function02()
    {

        Debug.Log("调用了第二个方法:非静态的,--继承Mono--");
    }
}
```

情景2,C#脚本不继承 MonoBehaviour,这种情况需要先调用构造函数,才可以调用非静态的方法和变量,测试代码如下:

```
using UnityEngine;

public class TestCall02  {
    public static string str01 = "这是测试脚本里的第一个变量:静态的";
    public string str02 = "这是测试脚本里的第二个变量:非静态的";


    //public static void AddCom(GameObject go)
    //{
```

```
    //    go.AddComponent<TestCall>();
    //}

    public static void Function01()
    {

        Debug.Log("调用了第一个方法:静态的");
    }

    public void Function02()
    {

        Debug.Log("调用了第二个方法:非静态的");
    }
}
```

2个脚本创建之后,我们需要在CustomSettings.cs脚本中的customTypeList对象里注册以上添加的2个脚本.具体代码为:(这一步十分重要,一定不要忘记添加)

```
_GT(typeof(TestCall)),
_GT(typeof(TestCall02))
```

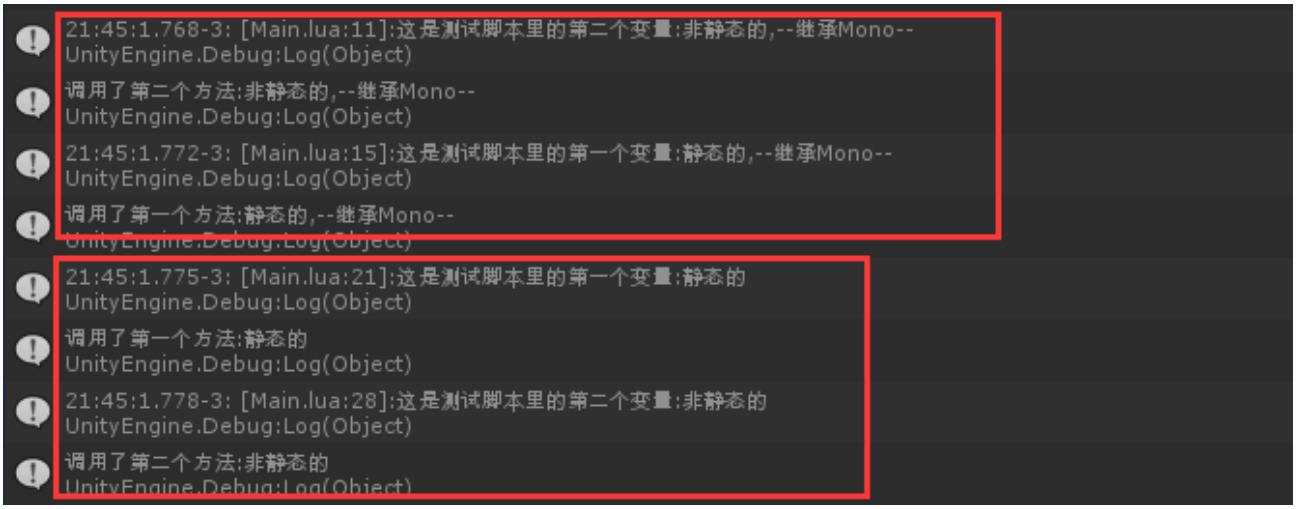
接下来我们可以在Main.lua脚本里的Main方法进行调试,测试代码如下:

```
--主入口函数。从这里开始lua逻辑
function Main()

    ------------------继承MonoBehaviour对象的调用测试-----------------------
    --继承MonoBehaviour的,需要将脚本先挂载到对象上
    go = UnityEngine.GameObject('go')
    --添加组件
    TestCall.AddCom(go);
    t=go:GetComponent("TestCall");
    --调用非静态的对象
    print(t.str02);
    t:Function02();

    --调用静态的变量
    print(t.str01);
    t.Function01();

```

```lua

    ------------------无继承MonoBehaviour对象的调用测试----------------------
    --调用静态的对象
    print(TestCall02.str01);
    TestCall02.Function01();

    --相当于C#的 t=new TestCall02();
    t02=TestCall02();

    --调用非静态的对象
    print(t02.str02);
    t02:Function02();

end
```

添加测试的lua代码之后,我们需要点击Lua菜单-Clear wrap files清除相关的包装文件,等待执行完毕之后,点击Generate All进行重新生成.再进行buidl android resources.

我们要在本地进行测试,还需要打开AppConst.cs脚本,把DebugMode=false,UpdateMode=false.保存回到工程,运行项目观察控制台,得到如下打印消息:



10.C#如何调用Lua代码?

任务:

创建一个lua脚本,其中包含自定义的:变量、方法、Table,然后创建一个C#脚本进行测试调用.

实操:

01.创建一个lua脚本,代码如下:

```lua
me="this is lua script";

function  StartCall()
    print(me);
    print("cull Lua function : TestCall")
end
StartCall();

function TestCallFunction(str)

    print("C# 调用了我");

    return str;
end

test={}
test.num=500
test.TestCall=TestCallFunction
```

02.创建C#调用脚本,代码如下:

```
using LuaInterface;
using System.Collections;
using UnityEngine;

public class CallLua : MonoBehaviour
{

    LuaState lua = null;
    LuaFunction func = null;
    public TextAsset text;

    void Start()
    {
        lua = new LuaState();
        lua.Start();
        lua.DoString(text.text);

        //调用lua对象的变量:
```

```
        var num = lua["test.num"];
        Debug.Log("lua里的变量num=:" + num);

        //也可以直接获取LuaTable:
        LuaTable t1 = (LuaTable)lua["test"];
        Debug.Log(t1["num"]);
        //如果使用LuaTable,记得释放内存
        t1.Dispose();

        func = lua.GetFunction("test.TestCall");
        //以下是2种方法调用形式,前者的lua对象占用的内存无法被自动释放;
        //但是可以调用func.Dispose()方法，释放掉垃圾内存，否则会造成内存泄漏。
        //而后者则不用关心GC释放的问题.
        if (func != null)
        {
            //有gc alloc
            object[] r = func.Call("男左");
            Debug.Log(r[0].ToString());
            //func.Dispose(); //如果这里调用,以下语句就不会执行了,func被释放掉.

            // no gc alloc
            string s = CallFunc();
            Debug.Log(s);
        }
    }

    string CallFunc()
    {
        func.BeginPCall();
        //给方法传参
        func.Push("女右");
        //通过func.Checkxxx()方法获得lua脚本里的返回值.
        string s = func.CheckString();
        //运行
        func.PCall();
        func.EndPCall();
        return s;
    }
}
```

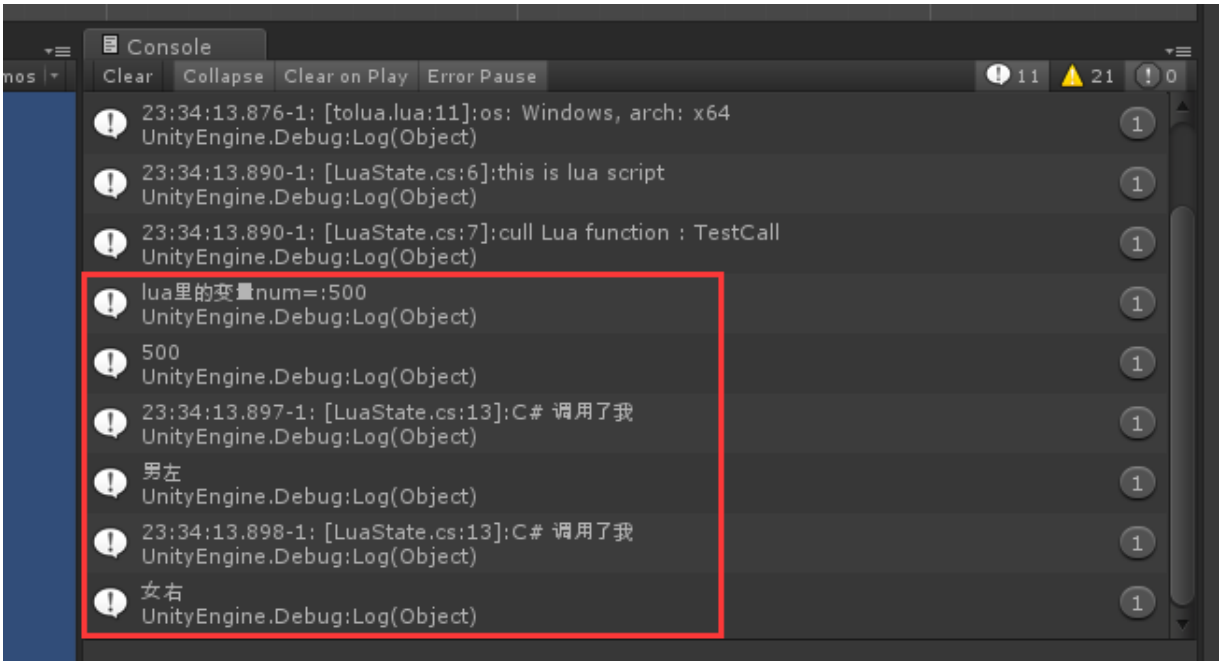
03.新创建一个场景,将以上C#脚本挂载到相机或者场景任一对象上.

04.将以上创建的lua脚本,扩展名加上.txt,然后指定给与CallLua组件的TextAsset对象.

05.我们需要点击Lua菜单-Clear wrap files清除相关的包装文件,等待执行完毕之后,点击Generate

All进行重新生成.再进行buidl android resources.

06我们要在本地进行测试,还需要打开AppConst.cs脚本,把DebugMode=false,UpdateMode=false.保存回到工程,运行项目观察控制台,依次得到如下打印出来的消息:



总结:

01.我们主要是通过LuaState、LuaFunction、LuaTable获得lua脚本里的变量、方法、Table.

02.执行lua代码则通过LuaState的DoString方法进行执行,需要传递对应的字符串,或者可以通过LuaState的DoFile执行lua脚本,需要传递脚本的路径.根据个人习惯进行使用即可.

03.通过LuaState对象GetFunction,传递lua脚本里对应的方法(字符串)形式,即可返回对应的LuaFunction对象.得到该对象就可以进行传参调用,具体参考以上02步骤的代码以及注释.

04.LuaTable使用的时候,要注意手动释放内存.所以不太推荐使用.但实际项目根据设计以及需求来开发就行了,没有绝对的规范.
