个人资料

房燕良

+加关注 发私信

博客专家

访问： 125189次
积分： 1743
等级： BLOG 4
排名： 第12304名

原创： 27篇 转载： 0篇
译文： 1篇 评论： 160条

原 在Unity中使用Lua脚本：语言层和游戏逻辑粘合层处理

分类： Unity3D 2015-04-11 11:38 2513人阅读 评论(5) 收藏 举报

游戏 脚本 unity3d lua

目录(?) [+]

前言：为什么要用Lua

首先要说，所有编程语言[illegible]的还是C#，VisualStudio+C#，只能说太舒服了。所以说，为什么非要在Unity里面用Lua呢？可[illegible]疼。。。。。另外还有一些次要原因：

- 方便做功能的热更[illegible]
- Lua语言的深度和广度都不大，易学易用，可以降低项目成本。

C#与Lua互相调用的方案

坦白来将，我并没有对现在C#与Lua互相调用的所有库进行一个仔细的调研，大概搜了一下，找到这样几个：

1. slua：https://github.com/pangweiwei/slua
2. Nlua：http://nlua.org/
3. UniLua：https://github.com/xebecnan/UniLua
4. uLua插件

以上这些方案的具体内容，不是本文的重点，这里就不说了，感兴趣的同学，点开自己去看就行了。

最后我选用了uLua，主要原因是：uLua方案比较成熟，它并没有太多自己的代码，主要是把LuaInterface和Lua解释器整合了一下，都是比较成熟的代码，相对会稳定一些。另外，个人很欣赏LuaInterface这个库。接下来我们就看一下uLua。：）

uLua的基本使用

uLua插件的使用非常简单，基本上看一下他自带的几个例子就明白了。

游戏逻辑粘合层设计

uLua插件解决了语言层面的问题：C#与LUA两种语言代码互相调用，以及参数传递等相关的一系列底层问题。而我们游戏逻辑开发中，到底如何使用LUA是上层的一个问题。下面给出我摸索的一个方案，个人认为：够简单，够清晰，是很薄很薄的一层，不可能更薄了。

使用几个LuaState？

曾经看过一个网友的方案，每次运行脚本就new一个LuaState，个人认为这种方案十分不妥。整个游戏的Lua代码应该运行在一个LuaState之上，原因有二：

1. 运行在同一LuaState的Lua代码才能互相调用啊。相信一个游戏总会有一定的代码量的，如果不同的lua文件之中的代码，完全独立运行，不能互相调用或者互相调用很麻烦，则游戏逻辑组织平添很多障碍；
2. 混合语言编程中原则之一就是：尽量减少代码执行的语言环境切换，因为这个的代价往往比代码字面上看上去要高很多。我的目标是：既然用了Lua，就尽量把UI事件响应等游戏上层逻辑放到Lua代码中编写。

基于以上原因，我觉得游戏的Lua代码全都跑在一个LuaState之上。这也是本文方案的基础。

实现LuaComponent

首先说一下我的目标：

- 既然C#对于Unity来说是脚本层了，那么Lua应该和C#脚本代码具有相同的逻辑地位；
- Lua整合的代码应该很少，应尽量保持简单；

基于以上的目标，我实现了LuaComponet类，它的实现类似MonoBehavior，只不过我们没有C++源代码，只能由C#层的MonoBehavior来转发一下调用。这样，我们的Lua代码的实现方式就是写和写一个C#脚本组件完全一致了，可以说达到了和引擎天衣无缝的整合。：）OK，先上代码！

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

/// <summary>
/// Lua组件 - 它调用的Lua脚本可以实现类似MonoBehaviour派生类的功能
/// </summary>
[AddComponentMenu("Lua/LuaComponent")]
public class LuaComponent : MonoBehaviour
{
    private static LuaState s_luaState; // 全局的Lua虚拟机

    [Tooltip("绑定的LUA脚本路径")]
    public TextAsset m_luaScript;

    public LuaTable LuaModule
    {
        get;
        private set;
    }
    LuaFunction m_luaUpdate;    // Lua实现的Update函数，可能为null

    /// <summary>
    /// 找到游戏对象上绑定的LUA组件（Module对象）
    /// </summary>
    public static LuaTable GetLuaComponent(GameObject go)
    {
        LuaComponent luaComp = go.GetComponent<luacomponent>();
        if (luaComp == null)
            return null;
        return luaComp.LuaModule;
    }

    /// <summary>
    /// 向一个GameObject添加一个LUA组件
    /// </summary>
    public static LuaTable AddLuaComponent(GameObject go, TextAsset luaFile)
    {
        LuaComponent luaComp = go.AddComponent<luacomponent>();
        luaComp.Initilize(luaFile);  // 手动调用脚本运行，以取得LuaTable返回值
        return luaComp.LuaModule;
    }

    /// <summary>
    /// 提供给外部手动执行LUA脚本的接口
    /// </summary>
    public void Initilize(TextAsset luaFile)
    {
```

```
        m_luaScript = luaFile;
        RunLuaFile(luaFile);

        //-- 取得常用的函数回调
        if (this.LuaModule != null)
        {
            m_luaUpdate = this.LuaModule["Update"] as LuaFunction;
        }
    }

    /// <summary>
    /// 调用Lua虚拟机，执行一个脚本文件
    /// </summary>
    void RunLuaFile(TextAsset luaFile)
    {
        if (luaFile == null || string.IsNullOrEmpty(luaFile.text))
            return;

        if (s_luaState == null)
            s_luaState = new LuaState();

        object[] luaRet = s_luaState.DoString(luaFile.text, luaFile.name, null);
        if (luaRet != null && luaRet.Length >= 1)
        {
            // 约定：第一个返回的Table对象作为Lua模块
            this.LuaModule = luaRet[0] as LuaTable;
        }
        else
        {
            Debug.LogError("Lua脚本没有返回Table对象：" + luaFile.name);
        }
    }

    // MonoBehaviour callback
    void Awake()
    {
        RunLuaFile(m_luaScript);
        CallLuaFunction("Awake", this.LuaModule, this.gameObject);
    }

    // MonoBehaviour callback
    void Start()
    {
        CallLuaFunction("Start", this.LuaModule, this.gameObject);
    }

    // MonoBehaviour callback
    void Update()
    {
        if (m_luaUpdate != null)
            m_luaUpdate.Call(this.LuaModule, this.gameObject);
    }

    /// <summary>
    /// 调用一个Lua组件中的函数
    /// </summary>
    void CallLuaFunction(string funcName, params object[] args)
    {
        if (this.LuaModule == null)
            return;

        LuaFunction func = this.LuaModule[funcName] as LuaFunction;
        if (func != null)
            func.Call(args);
    }
}


</luacomponent></luacomponent>
```

这段代码非常简单，实现以下几个功能点：

- 管理一个全局的LuaState；
- 负责将MonoBehavior的调用转发到相应的LUA函数；
- 提供了**GetComponent()**、**AddComponent()**对应的**LUA**脚本版本接口；这点非常重要。

LUA代码约定

为了很好的和LuaComponent协作，Lua脚本需要遵循一些约定：

- LUA脚本应该返回一个Table，可以是LUA的Module，也可以是任何的Table对象；
- 返回的Table对象应该含有MonoBehaviour相应的回调函数；

例如：

```
[plain]
01. require "EngineMain"
02.
03. local demoComponent = {}
04.
05. function demoComponent:Awake( gameObject )
06.     Debug.Log(gameObject.name.."Awake")
07. end
08.
09. return demoComponent
```

LuaComponent回调函数中，主动将GameObject对象作为参数传递给Lua层，以方便其进行相应的处理。

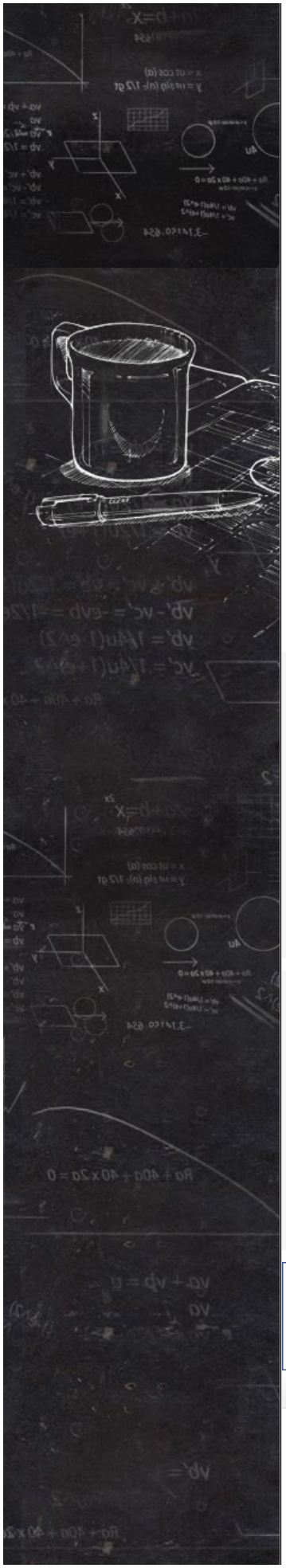
Lua组件之间的互相调用（在**Lua**代码中）

基于以上结构，就很容易实现Lua组件之间的互相调用。在Demo工程中，有一个“Sphere”对象，绑定了如下脚本：

```
[plain]
01. require "EngineMain"
02.
03. local sphereComponent = {}
04.
05. sphereComponent.text = "Hello World"
06.
07. function sphereComponent:Awake( gameObject )
08.     Debug.Log(gameObject.name.."Awake")
09. end
10.
11. return sphereComponent
```

还有另外一个“Cube”对象，绑定了如下脚本，用来演示调用上面这个Lua组件的成员：

```
[plain]
01. require "EngineMain"
02.
03. local demoComponent = {}
04.
05. function demoComponent:Awake( gameObject )
06.     Debug.Log(gameObject.name.."Awake")
07. end
08.
09. function demoComponent:Start( gameObject )
10.     Debug.Log(gameObject.name.."Start")
11.
12.     --演示LuaComponent代码互相调用
13.     local sphereGO = GameObject.Find("Sphere")
14.     local sphereLuaComp = LuaComponent.GetLuaComponent(sphereGO)
15.     Debug.log("Sphere.LuaDemoB:"..sphereLuaComp.text)
16.
```



```
17.  end
18.
19.  return demoComponent
```

完整版DEMO下载地址：

百度网盘链接: http://pan.baidu.com/s/1nt1eGPV 密码: 3g7b

最后，顺带总结一下：在设计上次游戏逻辑框架时，比较好的思路是：在透彻的理解Unity自身架构的前提下，在其架构下进行下一层设计，而不是想一种新的框架。因为Unity本身就是一个框架。

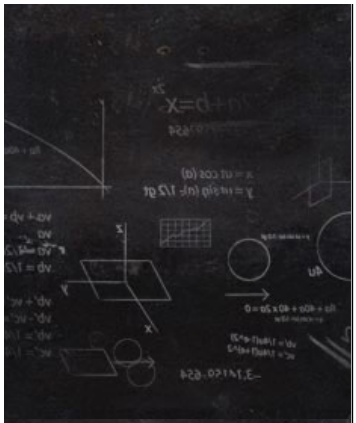
上一篇 分享一个小工具：UnityRemoteLog
下一篇 从Unreal Engine 3到Unreal Engine 4

查看评论

3楼 woshihuo12 2015-07-14 15:33发表

赞赞赞。

游戏热更新还是有很大用处的。可以解决一些游戏上线后临时出现的卡死 或者 其他重大bug，也可以调整游戏的一些逻辑更符合游戏运营的需要。



2楼 wang0312_ 2015-04-21 13:57发表

有没有办法动态的将Sphere传递给Cube的lua脚本，也可能是其它对象，而不使用GameObject.Find呢？

Re: 房燕良 2015-04-23 09:49发表

回复wang0312_：当然有可能了，你自己写一个方法就好。

1楼 秦元培 2015-04-11 13:17发表

以前试着把Unity的一部分功能封装起来然后交给Lua处理，结果发现工程量好大，而且有些方法还调用不了，这样好多了，可是有一个地方还不明白，找时间问下你

Re: 房燕良 2015-04-11 14:05发表

回复秦元培：LuaInterface已经帮你解决了LUA调用C#的问题，所以就不用“把Unity的一部分功能封装起来然后交给Lua处理”了。

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker
OpenStack VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC
WAP jQuery BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML
LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra
CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App
SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP
HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap