**XLua使用事项**

本文主要介绍Xlua的编码风格和使用限制, 在实际编码时遇到更详细的疑问，可查询XLua文档。

代码颜色：C# Lua

Lua访问有命名空间的对象/静态方法

local testM = CS.EIGame.CSTest

local testO = testM ()

testO:SayHello()

Lua访问无命名空间的对象方法

local testM = CS. CSTest

local testO = testM (“id”, 10)

testO:DoSomething(‘hello’)

--测试ref

local input = ' hello lua'

local outputValue = testO:DoSomething(input)

print(outputValue)

local outValue1, outValue2 = luaO2:DoSomething(inputValue)

print(outValue1)

print(outValue2)

--测试out

inputValue = ' hello lua '

outputValue = luaO2:SayHelloWithOutParam(inputValue)

print(outputValue)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NGUI\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

local label = CS.UILabel()

label.text = 'NGUI Test.'

print(label.text)

luaO3.onClick = function(obj)

print('hello lua')

print(obj)

end

luaO3.onClick('lua01')

**# Lua & C#语法**

**协程**

yield return WWW;

coroutine.yield(WWW);

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

yield break;

break;

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

yield return 0;

coroutine.yield(0);

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

yield return StartCoroutine (CreateLobbyCom());

coroutine.yield(this:StartCoroutine(this:CreateLobbyCom()));

**实例化**

SubTypeAndParam param = new SubTypeAndParam();

local param; param = SubTypeAndParam() ;

**取值 & 赋值**

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

静态

CS.EightGame.Logic.AdventureManager.\_newOpenLevelId = 0;

latestAreaId = CS.EightGame.Logic.AdventureManager.\_latestLevelId;

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

非静态

this.\_navigationNode = NavigationUINode() ;

i < obj\_len(this.\_subEINodeArray)

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

数组取值

ugCfg.donecondition[i].key

DictGetValue(ugCfg.donecondition, i).key

ugCfg.donecondition[“name”].key

DictGetValue(ugCfg.donecondition, ”name”).key

#DictGetValue为自定义的封装函数，后面说明。

**AddComponent & BindComponent**

com = go:AddComponent(typeof(LobbyViewCom));

this:BindComponent(com)

**事件监听**

this.entity:AddEventListener("LOGIN", function(e)

this:LoginAccount(e)

end

);

**委托 & 回调**

所有委托回调传function() … end即可。

this.Anim:SetFinishCallback(function()

this:OnLeaveLogin()

end

);

**创建类型**

List<string> Lst = List<string>();

local lst = CS.System.Collections.Generic["List`1[System.String]"]()

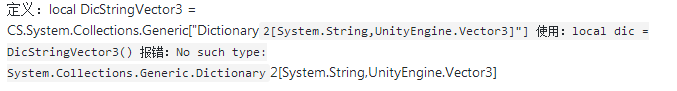
------------------------------

Dictionary<string, string> dict = new Dictionary<string, string>();

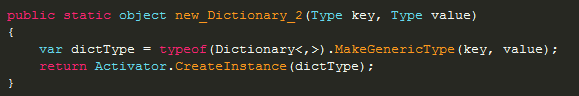
local dict = CS.System.Collections.Generic["Dictionary`2[System.String,System.String]"]

**但是有坑的是，除mscorlib，Assembly-CSharp程序集之外的类型都不支持，**

**需要用 C#的反射来创建，如：**



**C#的反射创建方式：**



**#XLua的约束**

1、不支持静态构造函数。

2、不支持在子类override函数通过base调用父类实现。

3、目前只支持Assets下代码的热补丁，不支持引擎，c#系统库的热补丁。

4、不支持非约束的泛型，如：

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

LogicStatic.Get< sd\_commonai > ( \_fighter.fighterInfo.commonAiScript );

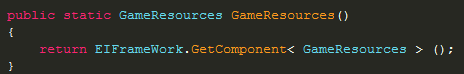
需改为：

LogicStatic.Get (typeof(sd\_commonai), \_fighter.fighterInfo.commonAiScript );

(即重载方法，添加Type参数进行泛型约束。)

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

或封装成接口使用，如：



**#XLua和 C#的交互**

正如上面所说，lua、c#混合编写，存在一些问题：

1. **类型数量、语法上差异和框架本身导致的约束：**

对于这一点，建议梳理出这些约束点，并尽量在c#提供相关的接口，供lua调用。

2、**类型在传递过程中，存在不确定性：**

当前类型可能是c#类型userdata；

也可能是lua原生类型；

又可能c# string类型传到lua后，变成lua原生string类型；

或者其他…

对于这一点，可以在lua层封装一层进行统一处理，如string、取值、foreach、nil判断等，公用方法可定义在/core/Global.lua

**#Lua编写规范**

原生Lua编写规范

。。。。