Slua User Guide

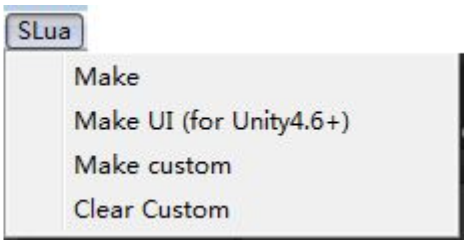
Siney/Pang weiwei
siney@yeah.net

1.Introduction

SLua is library for binding lua intreface for unity, it don't use reflection any more, but generate static binding wrap code. So Slua is very fast and gc-alloc is very small.
Slua is open source software, you can get latest version from
https://github.com/pangweiwei/slua.

2.Install

Download package and expand it to your Assets folder, waiting for Unity compiling, you will see a Slua Menu:



Click "Make" to generate Unity.Engine interface, you should generate UnityEngine interface at least.
Click "Make UI" to generate Unity.Engine.UI interface.
Click "Make custom" to generate your custom class(see below)
Click "Clear custom" to clear all custom interface file.

3.Example for tutor

Now you can run below lua code or open main project in example folder

```lua
-- import
import "UnityEngine"
function main()

    -- create gameobject
    local cube =
GameObject.CreatePrimitive(UnityEngine.PrimitiveType.Cube)

    -- find gameobject
    local go = GameObject.Find("Canvas/Button")

    -- get component by type name
    local btn = go:GetComponent("Button")

    -- add event listener
    btn.onClick:AddListener(function()
        local go = GameObject.Find("Canvas/Text")
        local label = go:GetComponent("Text")
        label.text="hello world"
    end)

    -- use vector3
    local pos = Vector3(10,10,10)+Vector3(1,1,1)
    cube.transform.position = pos
end
```

UnityEngine is a table in _G(global table) for Lua, you use import "UnityEngine" will copy all sub tables from UnityEngine to _G, so you can access sub types of UnityEngine, you can use UnityEngine.GameObject directly if not import it.

4.Export your custom class

Add class attribute [CustomClassAttribute] to your class, like:

```
[CustomLuaClassAttribute]
public class Custom : MonoBehaviour {
        ...
}
```

All public function/field/property/constructor in "Custom" class will generate wrap code automatically, if you don't want to export some function/field/property, can add [DoNotToLua] attribute, like:

```
[DoNotToLua]
static public void dontexport()
{
}
[DoNotToLua]
public int a;
[DoNotToLua]
```

```
public int A
{
    get
    {
        return a;
    }
}
```

Slua support overloaded methods, for your class(Not come from UnityEngine) is not recommended. Overloaded methods will cost more CPU time for searching proper method to call.

5.Delegate

Slua full support delegate(can be used in iOS), you can pass in lua function directly, if theis is a c# delegate like:

```
public delegate bool GetBundleInfoDelegate(string path, out string url);
public GetBundleInfoDelegate d;
```

In lua side, code like:

```
xxx.d=function(path)
    return true,url
end
```

Notice out modifier, you should return more value for every out parameter.

Slua also support +=/-= operator for delegate, like:

```
h.d={"+=",xxxDelegate}
h.d={"-=",xxxDelegate}
```

System.Action<>/System.Func<> also support, slua will flatten generic arguments.

6.Coroutine

Slua full support Unity’s yieldable methods, see code like below:

```
    import “UnityEngine”
    -- use coroutine
    local c=coroutine.create(function()
        print "coroutine start"
        Yield(WaitForSeconds(2))
        print "coroutine WaitForSeconds 2"

        local www = WWW("http://www.sineysoft.com")
        Yield(www)
        print(www.bytes)
        print(#www.bytes)
    end)
    coroutine.resume(c)
```

First of all, we use lua coroutine to wrap function, second use Yield function(import from UnityEngine) to yield any yieldable object.