2.XLua 热补丁

使用 XLua 来热更新的项目，原始项目在开发过程中全部用 C#编写，不需要写 Lua 代码，能大大提高项目的开发效率，只有当项中的功能需要热更新的时候，才需要编写 Lua 代码。

原项目出现了问题，写 Lua 代码进行修补，这种方式叫做“热补丁[HotFix]”。

3.热补丁简单演示

1.HotFix 特性标签

在使用 C#语言开发项目时，需要后续进行“热补丁修复”的类，需要在类的头部添加一个特性标签：`[Hotfix]`，表示该类可以被 XLua 热修复。

2.Hotfix 语法

```
xlua.hotfix(CS.类名, '方法名', lua 方法)
```

说明：

这个是 lua 代码结构，需要使用 Lua 虚拟机对象中的 DoString 方法执行。

含义就是：某个类中的某个方法，你用 lua 方法进行修复。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using XLua;
[Hotfix]
public class MyXLua : MonoBehaviour
{

    private LuaEnv luaEnv;
```

```
    private string luaHotScripts = @"
    xlua.hotfix(CS.MyXLua,'Hello',
    function()
        print('lua Hello~~')
    end )

    xlua.hotfix(CS.MyXLua,'Add',
    function(self,a,b)
        print('lua Add:'..(a..b))
    end)
";
    // Use this for initialization
    void Start()
    {
        luaEnv = new LuaEnv();
    }



    public void Hello()
    {
        Debug.Log("Hellow~~~");
    }

    public void Add(int a,int b)
    {
        Debug.Log("C#ADD:" + (a + b));
    }
    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.A))
        {
            Hello();
            Add(1,3);
        }
```

```
        if (Input.GetKeyDown(KeyCode.Space))
        {
            luaEnv.DoString(luaHotScripts);
        }
    }
}
```

加载外部场景修复

1.加载外部文件修复

1.基础介绍

使用 XLua 进行热更新的项目，在项目启动的时候，会加载执行一个外部的 Lua 文件，对当前项目中的 C#类或者方法进行热修复，修复完毕后，再进入游戏，这样就完成了项目的热更新。

2.操作步骤

①将 Lua 代码定义到独立的 Lua 代码文件中；

②创建一个独立的游戏物体，配套一个独立的 C#脚本，在这个脚本内定义 Lua 虚拟机，并且由该脚本完成 Lua 的热更新逻辑；

2.多场景跳转修复

1.基础介绍

在正式项目开发过程中，一个项目往往是由多个场景构成，每个场景内都会有很多代码，对其他场景内代码的修复，也是在最开始的场景内完成的。

[比如：RPG 手游，往往是在最开始的资源加载界面，完成热更新修复。]

2.操作步骤

①创建一个新的场景，编写场景的逻辑代码；

②在开始场景内编写场景的跳转逻辑，进行初步测试；

③编写 Lua 热修复代码，完成跨场景代码修复。

1.有参游戏逻辑修复

1.需求分析

首先使用 C#语言编写游戏逻辑，读取一个 Cube 预制体，然后定义一个方法，

在这个方法内，循环生成一堵墙壁，这个方法有一个参数，用于传递预制体。

然后使用 Lua 语言对这个方法进行修复。

2.操作步骤

①使用 C#完成代码逻辑；

②在 Lua 文件中编写修复逻辑；

③使用 HotFixManager 完成 Lua 热更新。

2.Lua 访问 C#脚本内的字段

方法 1：

可以在 lua 代码内通过“self.字段名”进行访问，但是这样有一个前提，就是该字段必须是 public 修饰的，private 修饰的访问不到。

但是这样有一个弊端，破坏了 C#语言本身的“封装性”。

方法 2：

在 lua 语言中，使用代码获取 C#类中 private 成员的访问权，代码如下：

```
xlua.private_accessible(CS.类名)
```

加上这句话，就可以在 Lua 脚本中访问到 C#类当中的私有成员，同时不会破坏 C#原有的封装性和逻辑关系。