English　　FAQ
简体中文　　常见问题解答

C#Like免费版简介

v1.1 更新日期2023-02-27

C#Like是Unity的热更方案,使用C#语言,无缝将传统UnityC#脚本变成可热更的方案.
C#Like免费版是C#Like的阉割版,你可以在任何地方(含Unity商店)下载它,这是一个免费资源.
如果你不符合Unity个人版的条件,你不能将C#Like免费版用于商业用途.
对于拥有Unity个人许可的用户,可以使用C#轻松构建热更新项目;
对于其他的用户,可以通过试用C#Like免费版 来了解C#Like是否值得购买.
C#Like免费版和C#Like目录结构和使用流程是一样的,升级非常方便.

无尽感激开源项目C#Light

下面是免费版和完整版的示范导出WebGL平台的Demo网页 (欢迎导出C#免费版验证热更流程):
C#LikeFree演示　　C#Like演示

功能概要对比

功能	C#Light	C#Like免费版	C#Like
委托	支持	支持	支持
Lambda	支持	支持	支持
面向对象	类仅能继承接口	多了支持partial	多了构造函数(支持this/base);析构函数;类继承;虚函数
运算表达式	+ - * / % += -= *= /= %= > >= < <= != == && || ! ++ --(仅支持i++不支持++i) ?: is as	多了完全支持++ --	多了位运算 & | ~ ^ &= |= ^= << >> <<= >>=
关键字	this	this typeof	this base sizeof typeof unsafe $ @ #pragma #warning #error
命名空间 using	不支持,有名无实,导致每种在热更代码里使用的所有类型必须先非热更注册,非常不便且容易遗漏	完整的命名空间功能,**无需事先注册类型**就能直接使用,使用方便;using指令/别名/static;	多了using语句
异常处理	throw	同C#Light	多了try-catch-finally
类型	var void bool float double char string byte sbyte int uint short ushort long ulong null	同C#Light	多了支持可空运算?及配套的合并运算?. ??
get/set访问器	仅支持自动实现的get/set	同C#Light	多了可以自定义实现get/set
循环语句	for foreach continue break if-else return while do-while	同C#Light	多了switch-case-default
调试	能打印报错语句	在调试模式下可以使用Visual Studio**断点/步进调试**,与正常C#一模一样;热更代码模式下当脚本出错可以打印出堆栈数据(文件名/函数名/第几行)	同C#Like免费版,但支持更多C#特性,大幅减少调试模式和热更代码模式差异
编译脚本	在运行时编译,视乎你的代码量多少可能会花费几秒甚至超过十几秒,即使已经预编译为Token了	所有编译过程都在编辑器里完成,保存为二进制文件,**运行时加载时间几乎可以忽略**,虽然编译时间也基本一样(首次编译甚至花费更多时间建立缓存数据),但加载时候给玩家极好的体验	同C#Like免费版
MonoBehaviour	不支持	LikeBehaviour仿照MonoBehaviour,让热更脚本直接按MonoBehaviour的方式写代码,热更脚本就像是继承MonoBehaviour一样	多了**完美支持协程**
多线程	不支持	支持	支持且多了lock语法
注释	仅//	仅//	//和/**/
宏定义和区域	不支持	不支持	#if #elif #else #endif #region #endregion
枚举	不支持	不支持	支持
参数修饰符	不支持	不支持	**支持ref out in params**
函数重载	不支持	不支持	支持
默认参数	不支持	不支持	支持
CSV	不支持	KissCSV	同C#Like免费版
JSON	无	KissJSON	同C#Like免费版
Socket/WebSocket	无	超简洁的可热更的接口,提供KissServerFramework(最简洁易用的IOCP服务器框架,用户逻辑单线程,后台数据库多线程,面向对象,操作极简,包含WebSocket/Socket/HTTP/MySQL,你不会用到SQL的,只需定义数据库表结构,即可使用数据且自动和客户端和数据库三者间同步数据)	同C#Like免费版

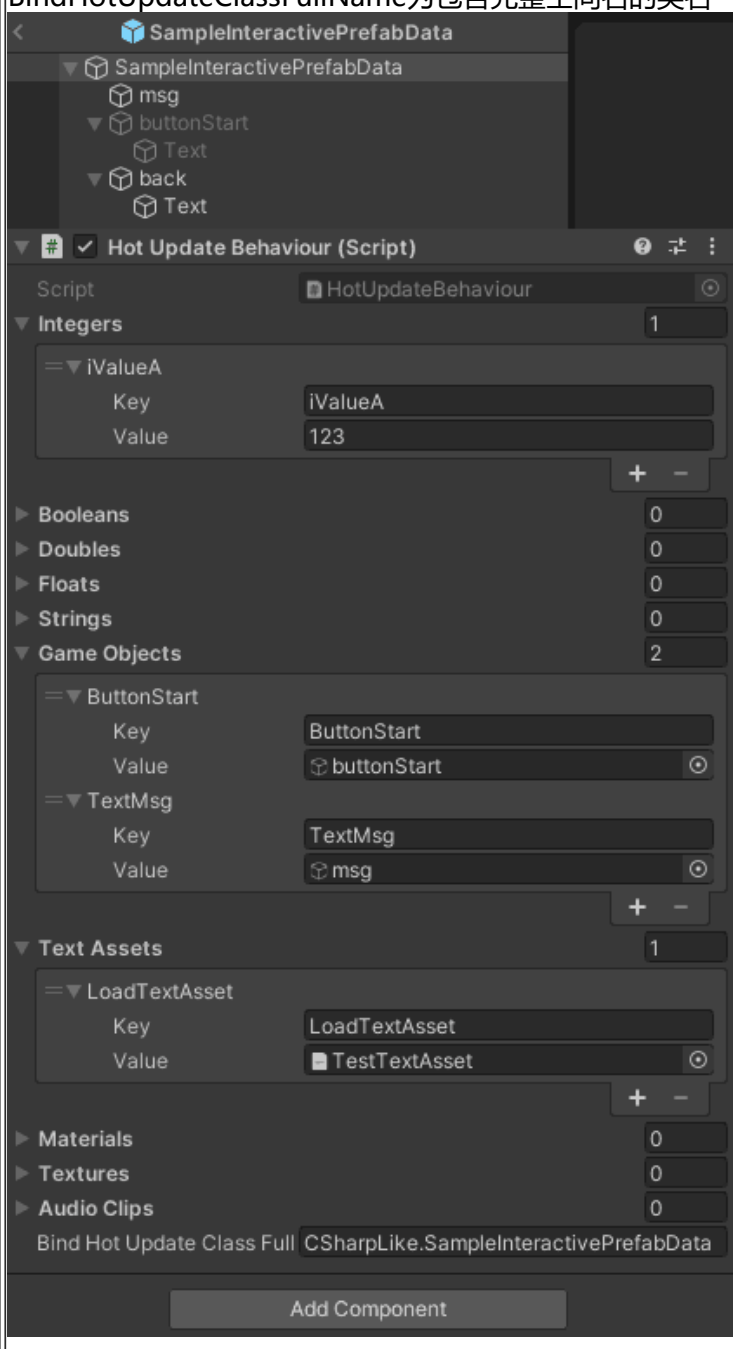
C#Like免费版快速了解

传统unity代码	C#Like热更代码
传统非热更的代码.我们定义一个最简单的类,继承于MonoBehaviour,调用最常用的几个生命周期函数	这里我们写一个跟左边传统unity代码一模一样功能的热更代码,大家可以对比一下两者的区别

```
using UnityEngine;
using System;

namespace CSharpLike
{
    /// <summary>
    /// 最简单的非热更Hello World例子
    /// 示范最简单的Unity生命周期函数
    /// 和协程的简单用法
    /// </summary>
    public class SampleHelloWorld : MonoBehaviour
    {
        void Awake()
        {
            gameObject.name = "I'm MonoBehaviour";
            Debug.Log("Awake");
        }
        void OnEnable()
        {
            Debug.Log("OnEnable");
        }
        IEnumerator Start()
        {
            Debug.Log("Start:这里返回类型为IEnumerator表示为协程模式");
            Debug.LogError("Start:step0: " + DateTime.Now);
            yield return new WaitForSeconds(2f);
            Debug.LogError("Start:step1: " + DateTime.Now);
            yield return null;
            Debug.LogError("Start:step2: " + DateTime.Now);
            yield return StartCoroutine("SubCoroutine", "test", 123, 321f);
            Debug.LogError("Start:step3: " + DateTime.Now);
        }
        IEnumerator SubCoroutine(string str, int iValue, float fValue)
        {
            Debug.LogError("SubCoroutine:
("" + str + "","" + iValue + "","" + fValue + "") "" + DateTime.Now);
            //这里只示范 WaitForSeconds
            yield return new WaitForSeconds(2f);
            Debug.LogError("SubCoroutine:end " + DateTime.Now);
        }
        void FixedUpdate()
        {
            Debug.Log("FixedUpdate");
        }
        float angle = 0f;
        void Update()
        {
            //Debug.Log("LateUpdate");
            angle += Time.deltaTime * 50f;
            transform.localEulerAngles = new Vector3(0f, angle, 0f);
        }
        void LateUpdate()
        {
            Debug.Log("LateUpdate");
        }
        void OnGUI()
        {
            //Debug.Log("OnGUI");
            if (GUI.Button(new Rect(0, 0, 64, 64), "Back"))
            {
                SampleCSharpLike.instance.gameObject.SetActive(true);//回到上一层示范
                HotUpdateManager.Hide("Sample/SampleHelloWorld");//关掉自身
            }
        }
        void OnDisable()
        {
            Debug.Log("OnDisable");
        }
        void OnDestroy()
        {
            Debug.Log("OnDestroy");
        }
    }
}
```

```
using UnityEngine;
using System;

namespace CSharpLike
{
    /// <summary>
    /// 最简单的热更Hello World例子.
    /// 示范最简单的Unity生命周期函数(支持所有Behaviour所有事件,这里示范最常用的).
    /// 和模仿协程的简单用法.
    /// </summary>
    public class SampleHelloWorld : LikeBehaviour
    {
        void Awake()
        {
            //同MonoBehaviour的gameObject
            gameObject.name = "I'm LikeBehaviour";
            Debug.Log("Awake:和Unity MonoBehaviour的完全一样");
        }
        void OnEnable()
        {
            Debug.Log("OnEnable:和Unity MonoBehaviour的完全一样");
        }
        int stepStart = 0;
        void Start()
        {
            if (stepStart == 0)
            {
                Debug.Log("Start:和Unity MonoBehaviour的一样但不能使用协程");
                Debug.LogError("示范协程: 你不能在C#Like免费版里使用协程"+
                " (在完整版已支持).强烈推荐升级到完整版: " +
                "https://assetstore.unity.com/packages/slug/222256");
                // 我们提供以下方案给免费用户:
                // 协程: 使用 MemberCallDelay 模拟协程 (只能模拟
                // 'yield return new WaitForSeconds(float seconds);'
                // 和 'yield return null;').
                // 协程是Unity非常强大的功能,它能够让我们的代码逻辑变得更清晰整洁,
                // 你不去使用它实在太可惜了.
                Debug.LogError("Start:step0: " + DateTime.Now);
                stepStart++;
                MemberCallDelay("Start", 2f);
            }
            else if (stepStart == 1)
            {
                Debug.LogError("Start:step1: " + DateTime.Now);
                stepStart++;
                MemberCallDelay("Start");
            }
            else if (stepStart == 2)
            {
                Debug.LogError("Start:step2: " + DateTime.Now);
                stepStart++;
                SubCoroutine("test", 123, 321f);
            }
            else if (stepStart == 3)
            {
                Debug.LogError("Start:step3: " + DateTime.Now);
                stepStart = 0;
            }
        }
        int stepSubCoroutine = 0;
        void SubCoroutine(string str, int iValue, float fValue)
        {
            if (stepSubCoroutine == 0)
            {
                Debug.LogError("SubCoroutine:
("" + str + "","" + iValue + "","" + fValue + "") "" + DateTime.Now);
                stepSubCoroutine++;
                MemberCallDelay("SubCoroutine", str, iValue, fValue, 2f);
            }
            else if (stepSubCoroutine == 1)
            {
                Debug.LogError("SubCoroutine:end " + DateTime.Now);
                MemberCallDelay("Start");
                stepSubCoroutine = 0;
            }
        }
        void FixedUpdate()
        {
            Debug.Log("FixedUpdate:和Unity MonoBehaviour的完全一样");
        }
        float angle = 0f;
        void Update()
        {
            //Debug.Log("LateUpdate:和Unity MonoBehaviour的完全一样");
            angle += Time.deltaTime * 50f;
            transform.localEulerAngles = new Vector3(0f, angle, 0f);
        }
        void LateUpdate()
        {
            Debug.Log("LateUpdate:和Unity MonoBehaviour的完全一样");
        }
        void OnGUI()
        {
            //Debug.Log("OnGUI:和Unity MonoBehaviour的完全一样");
            if (GUI.Button(new Rect(0, 0, 64, 64), "Back"))
            {
                //返回上一层及关掉自身
                SampleCSharpLike.instance.gameObject.SetActive(true);
                HotUpdateManager.Hide("Sample/SampleHelloWorld");
            }
        }
        void OnDisable()
        {
            Debug.Log("OnDisable:和Unity MonoBehaviour的完全一样");
        }
        void OnDestroy()
        {
            Debug.Log("OnDestroy:和Unity MonoBehaviour的完全一样");
        }
    }
}

```

C#Like免费版如何和Unity交互(与完整版完全一致)

在预制体中设置	纯热更交互
如下图,示范如何获取各种数值类数据和绑定的组件. BindHotUpdateClassFullName为包含完整空间名的类名 	这个是热更脚本跟Unity交互数据 见下方代码


```cs
using System;
using UnityEngine;
using UnityEngine.UI;

namespace CSharpLike
{
    /// <summary>
    /// 热更脚本和Unity交互数据的例子
    /// </summary>
    public class ExampleInteractivePrefabData : LikeBehaviour
    {
        void Start()
        {
            //获取当前热更新脚本组件,该组件继承于MonoBehaviour,可看做MonoBehaviour
            Debug.Log("HotUpdateBehaviour=" + behaviour);
            //同MonoBehaviour的gameObject
            Debug.Log("gameObject.name=" + gameObject.name);
            //同MonoBehaviour的transform
            Debug.Log("transform.localPosition=" + transform.localPosition);

            //获取预制体里设置的数值类数据(int/bool/float/double/string),以字符串为键值
            //这里获取int数值为例子,在预制体中,我们设置一个键值'iValueA'的值为123
            //GetBoolean/GetDouble/GetFloat/GetString
            Debug.Log("integer value set in prefab:iValue=" + GetInt("iValueA"));
            //我们可以修改该数值,它会在编辑器里刷新.
            //你也可以运行中在编辑器里直接修改它.
            SetInt("iValueA", 321);//SetBoolean/SetDouble/SetFloat/SetString
            //这里再次调用一下该数值,看修改是否有效
            Debug.Log("integer value after modify:iValue=" + GetInt("iValueA"));

            //我们可以获取预制体里Unity的组件(GameObject/TextAsset/Material/Texture/AudioClip),
            //同样也是以字符串为键值.
            //这里获取GameObject为例子,在预制体中,我们设置了一个键值为'ButtonStart'的GameObject
            GameObject goStart = GetGameObject("ButtonStart");
            goStart.SetActive(true);//我们设置它为活动,在预制体中,我们设置隐藏的

            //我们这里示范获取预制体绑定的文本组件(UnityEngine.UI.Text)
            //你可以获取在'运行时'绑定在HotUpdateBehaviour的gameObjects组件.
            //你可以获取任何可以通过GameObject.GetComponent<T>()来获取的Unity组件.
            //例如Image/Button/Slider/MeshRenderer/...等等
            Text text = GetComponent<Text>("TextMsg");

            //这里示范获取绑定在预制体的'资源'(TextAsset/Material/Texture/AudioClip),
            //例如文本资源(TextAsset)这些在编辑器不能拖在GameObject列的.
            TextAsset ta = GetTextAsset("LoadTextAsset");//GetMaterial/GetTexture/GetAudioClip
            Debug.Log("Load TextAsset:" + ta.text);
            text.text = ta.text;//读取文本资源后,在前面绑定的文本组件里显示
        }
        void OnClick()//'ButtonStart'绑定的按钮响应
        {
            //点击按钮,在文本组件里显示当前时间
            GetComponent<Text>("TextMsg").text = DateTime.Now.ToString();
        }
    }
}
```

C#Like免费版无需事先注册类型(与完整版完全一致)

C#Light	C#Like
必须先在非热更脚本里注册各种类型,才能在热更脚本里使用,且无法使用using来区分同类名不同命名空间的类.	无需提前注册类型,直接使用.防止项目发布了,突然发现得用某个类却无法使用修改非热更代码就嗝屁了.

```cs
//需先在事先在非热更代码里加入
RegisterType(typeof(UnityEngine.Vector2), "Vector2");
//才可以在热更代码里使用
Vector2 vector = new Vector2();
//UnityEngine.Vector2 vector = new UnityEngine.Vector2();//完整名写法报错

//同类名不同命名空间的类在整个热更代码只能2选1
//RegisterType(typeof(UnityEngine.Random), "Random");
//RegisterType(typeof(System.Random), "Random");
```

```cs
//直接在热更代码使用
Vector2 vector = new Vector2();
UnityEngine.Vector2 vector2 = new UnityEngine.Vector2();
//同类名不同命名空间的类均可使用
int random1 = UnityEngine.Random.Range(0, 100);
int random2 = (new System.Random()).Next(100);
//或者使用别名
using Random = UnityEngine.Random;
int random3 = Random.Range(0, 100);
```

IL2CPP裁剪(与完整版完全一致)

类型裁剪	泛型处理
IL2CPP会将所有没用到的类型裁剪掉,很遗憾我们非热更的代码刚好是被裁剪的对象, 必须把你将要使用的模块设置为排除.Managed Stripping Level设置Low或Medium(不能选High). 在Unity安装目录Editor\Data\Managed\UnityEngine\里查看一下自己想要的dll,然后自行修改Assets\C#Like\link.xml这个文件	IL2CPP必须AOT,无法JIT来创建.所有的泛型必须在先在非热更代码使用一次才能非热更代码里使用,我们在编译代码的时候会自动把热更代码里所有需要注册的泛型自动生成Assets\C#Like\Runtime\AheadOfTime\AheadOfTime.cs,理论上你的产品上线时候,基本上可以把所有需要的泛型都注册了.下面是例子自动生成的文件(这个文件别手动修改)

```xml
<linker>
    <assembly fullname="System" preserve="all"/>
    <assembly fullname="mscorlib" preserve="all"/>
    <assembly fullname="Assembly-CSharp" preserve="all"/>
    <assembly fullname="UnityEngine" preserve="all"/>
    <assembly fullname="UnityEngine.CoreModule" preserve="all"/>
    <assembly fullname="UnityEngine.UIModule" preserve="all"/>
    <assembly fullname="UnityEngine.IMGUIModule" preserve="all"/>
    <assembly fullname="UnityEngine.InputModule" preserve="all"/>
</linker>
```

```cs
/*
 *          C#Like
 * Copyright © 2022 RongRong
 * It's automatic generate by CSharpLikeEditor,don't modify this file.
 */

namespace CSharpLike
{
    namespace Internal
    {
        /// <summary>
        /// Ahead-of-time compile for generic type in ScriptingBackend with IL2CPP
        /// </summary>
        public class AheadOfTime
        {
            public AheadOfTime()
            {
                new System.Collections.Generic.List<UnityEngine.Vector3>();
            }
        }
    }
}
```

```
23.
```

C#Like免费版使用流程(与完整版完全一致)

文件框架目录

```
C#Like
    |
    |--Documentation
    |    |--C#LikeManualCN.pdf                    //用户手册
    |
    |--Editor                                     //编辑器, 你可以忽略它
    |
    |--HotUpdateScripts                           //默认的热更脚本目录,这个目录放"热更"的脚本.
    |    |                                        //导出发布的时候你"可以"删掉这个目录所有内容(记得备份好哦)
    |    |--Sample                                //示例的热更脚本,仅示范用,你可以随时删掉它
    |    |    |--SampleInteractivePrefabData.cs   //热更脚本和Unity预制体交互数据的例子
    |    |    |--SampleHelloWorld.cs              //最简单的热更Hello World例子,示范最简单的Unity生命周期函数和协程的简单用法
    |    |    |--SampleC#                         //示例所支持的C#特性,建议看一下
    |    |    |--AircraftBattle                   //一个打飞机小游戏的热更代码示例
    |    |    |--NetObjects                       //客户端和服务器端之间的传输对象示例
    |    |    |--SampleSocket.cs                  //使用Socket/WebSocket和服务器交互的例子
    |    |
    |    |--SampleFullVersion.zip                 //完整版的示例热更脚本(仅在完整版可以使用),你可以查看发现免费版和完整版的区别
    |
    |--Resources
    |    |--Sample                                //示例用到的资源文件,仅示范用,你可以随时删掉它
    |
    |--Runtime                                    //运行时脚本,这个目录放"非热更"的脚本
    |    |--AheadOfTime                           //编辑器自动生成AOT脚本,应对IL2CPP裁剪,别动它
    |    |--Interaction                           //C#交互部分,你应该关注这个目录
    |    |    |--CLS_Environment.cs               //C#Light/C#Like免费版/C#Like的三者比较,和C#Light作者的声明
    |    |    |--CSL_Tools.cs                     //自定义最终导出二进制文件的加密解密方法
    |    |    |--HotUpdateBehaviour.cs            //热更脚本行为,与MonoBehaviour交互的载体,
    |    |    |                                   //仅包含Awake/OnEnable/Start/OnDisable/OnDestroy行为
    |    |    |--HotUpdateManager.cs              //热更脚本管理脚本
    |    |    |--LikeBehaviour.cs                 //热更脚本版的MonoBehaviour,如果在热更脚本中如果想跟预制体交互,就以这个作为基类.
    |    |    |                                   //你可以把它当作热更脚本里的MonoBehaviour
    |    |    |--MyCustomConfig.cs                //自定义配置
    |    |    |--MyHotUpdateManager.cs            //自定义热更脚本管理脚本
    |    |    |--HotUpdateBehaviourCustom         //你应该在自定义更多的HotUpdateBehaviour,这样你就包含可以更多
    |    |         |                              //除了'Awake/OnEnable/Start/OnDisable/OnDestroy'的行为,来适配你的需求
    |    |         |--HotUpdateBehaviourAll.cs             //包含MonoBehaviour"所有"行为的脚本,不建议使用,但这个是遗漏自定义合适行为的后备方案
    |    |         |--HotUpdateBehaviourApplication.cs     //额外包含OnApplicationFocus/OnApplicationPause/OnApplicationQuit行为
    |    |         |--HotUpdateBehaviourCollision.cs       //额外包含OnCollisionEnter/OnCollisionExit/OnCollisionStay行为
    |    |         |--HotUpdateBehaviourCollision2D.cs     //额外包含OnCollisionEnter2D/OnCollisionExit2D/OnCollisionStay2D行为
    |    |         |--HotUpdateBehaviourCommon.cs          //额外包含FixedUpdate/LateUpdate/Update/OnGUI行为
    |    |         |--HotUpdateBehaviourFixedUpdate.cs     //额外包含FixedUpdate行为
    |    |         |--HotUpdateBehaviourLateUpdate.cs      //额外包含LateUpdate行为
    |    |         |--HotUpdateBehaviourOnGUI.cs           //额外包含OnGUI行为
    |    |         |--HotUpdateBehaviourTrigger.cs         //额外包含OnTriggerEnter/OnTriggerExit/OnTriggerStay行为
    |    |         |--HotUpdateBehaviourTrigger2D.cs       //额外包含OnTriggerEnter2D/OnTriggerExit2D/OnTriggerStay2D行为
    |    |         |--HotUpdateBehaviourUpdate.cs          //额外包含Update行为
    |    |         |--HotUpdateBehaviourUpdateTrigger2D.cs //额外包含OnCollisionEnter2D/OnCollisionExit2D/OnTriggerStay2D/Update行为
    |    |
    |    |--Internal                             //C#Like内部实现代码,你可以忽略它
    |    |--KissCSV                              //读取CSV文件的库
    |    |--KissJson                             //为C#Like量身打造的JSON解析库,你唯一的选择
    |    |--Sample                               //示例用到的非热更脚本,仅示范用,你可以随时删掉它
    |         |--SampleCSharpLike.cs             //示范用的首个界面脚本,里面包含初始化C#Like和加载热更脚本文件和调起4个例子的预制体的按钮
    |         |--SampleNotHotUpdateScript.cs     //示范用的非热更代码
    |         |--MyAnchor.cs                     //示范用的锚点
    |         |--MyRoot.cs                       //示范用的根节点
    |
    |--Scenes
    |    |--SampleScene.unity                    //示例用到的场景文件,仅示范用,你可以随时删掉它
    |
    |--Plugins
         |--WebGL                                //WebGLC平台使用的WebSocket库.
```

下面是搭建整个框架(C#LikeFree+KissServerFramework)的过程,如果只看C#LikeFree部分,可以只看步骤6

```
1 搭建KissServerFramework服务器
    1.1 下载服务器源码,使用Visual Studio(我自己使用VS2019在.NET5)编译发布最终单个exe(如果嫌弃太麻烦,可以直接使用我编译的KissServerFramework.exe),例如放C盘,目录结构为:
        C:\KissServerFramework
            |
            |--CSV
            |  |--Item.csv                        //测试用的CSV文件
            |
            |--KissServerFramework.exe           //主程序,你选择自行编译或直接使用我编译好的
            |--KissServerFramework.json          //配置JSON文件
            |--kiss.sql                          //这个是对应数据库SQL文件,仅在步骤3里使用

    1.2 如果目标机器没有安装.NET5运行时库的需要去微软官网下载且安装:https://download.visualstudio.microsoft.com/download/pr/a0832b5a-6900-442b-af79-6ffddddd6ba4/e2df0b25dd851ee0b38a86947dd0e42e/dotnet-runtime-5.0.17-win-x64.exe
    1.3 如有需要则修改KissServerFramework.json里的WebSocket端口9000,Socket端口9001,HTTP端口9002,请确保这些端口没有已经被占用
    1.4 如果存在防火墙,则需要对外开放TCP端口9000~9002

2 搭建XAMPP
    2.1 XAMPP的官网下载 https://www.apachefriends.org/download.html
    2.2 一直"下一步"地安装XAMPP直至安装完毕(默认安装到C:/xampp)
    2.3 确认MySQL和Apache开着的, 你可以在XAMPP控制面板查看.
    2.4 如果存在防火墙,则需要对外开放TCP端口80

3 创建"kiss"数据库且导入"kiss.sql"
    3.1 执行批处理文件files/CreateDatabase.bat(默认没密码,要求输入密码直接按回车键), 如果XAMPP安装路径有变,请修改bat文件
    3.2 执行批处理文件files/ImportDatabase.bat(默认没密码,要求输入密码直接按回车键), 如果XAMPP安装路径有变,请修改bat文件

4 可选步骤:升级HTTP到HTTPS,升级WS到WSS,更换Socket的RSA证书
    4.1 准备SSL证书,可以到腾讯云( https://console.cloud.tencent.com/ssl )或"Let's Encrypt"( https://letsencrypt.org/ )申请免费的SSL证书或购买.我自己是腾讯云申请的免费SSL证书,下载Apache版证书和Nginx版证书备用.
    4.2 安装Nginx代理
        4.2.1 Nginx官网下载 http://nginx.org/en/download.html 一般我们下载里面的稳定版本,我们这里示范选nginx.Windows-1.22.1
        4.2.2 下载的回来的nginx-1.22.1.zip解压在C盘中
        4.2.3 复制"files/QuitNginx.bat" "files/RetartNginx.bat" "files/StartNginx.bat" "files/StopNginx.bat" 4个文件到Nginx目录下
        4.2.4 把"files/nginx.conf"替换"C:\nginx-1.22.1\conf\nginx.conf"
        4.2.5 把前面申请的Nginx版的证书解压到C:\nginx-1.22.1\conf\xxxx.com\目录内
            最后文件结构如下
            C:\nginx-1.22.1
                |
                |--conf
                |  |--xxxx.com                   //Nginx版的证书
                |  |   |--xxxx.com_bundle.crt
                |  |   |--xxxx.com.key
                |  |
                |  |--nginx.conf                 //配置文件
```

```
                |
                |--contrib
                |--docs
                |--html
                |--logs
                |--temp
                |--nginx.exe
                |
                |--QuitNginx.bat            //双击这个是退出Nginx
                |--RetartNginx.bat          //双击这个是重启Nginx,例如修改配置后用
                |--StartNginx.bat           //双击这个是启动Nginx,例如首次打开时候用
                |--StopNginx.bat            //双击这个是停止Nginx

        4.2.6 设置C:/nginx-1.22.1/conf/nginx.conf文件,把里面的xxxx.com替换成你实际域名,例如我自己的是csharplike.com替换xxxx.com
        4.2.7 如有需要修改nginx.conf里面的端口,里面设置WSS端口10000代理到WS端口9000,HTTPS端口10002代理到HTTP端口9002,如果有防火墙,请开启对于TCP端口(10000和10002)
        4.2.8 因为采用了WSS和HTTPS,请修改C:\KissServerFramework\KissServerFramework.json的网关端口:
            "serverInfos"节点内的"WebSocketURI": "wss://www.xxxx.com:10000"
            "serverInfos"节点内的"HttpURI": "https://www.xxxx.com:10002"
        4.2.9 最后记得执行StartNginx.bat启动Nginx,如果前面已经启动过可以执行RetartNginx.bat

    4.3 配置Apache的SSL证书
        4.2.1 把下载的Apache版的证书放到C:\xampp\apache\conf\xxxx.com\目录内
        4.2.2 如下修改C:\xampp\apache\conf\extra\httpd-ssl.conf
            SSLCertificateFile "C:/xampp/apache/conf/xxxx.com/xxxx.com.crt"
            SSLCertificateKeyFile "C:/xampp/apache/conf/xxxx.com/xxxx.com.key"
            SSLCertificateChainFile "C:/xampp/apache/conf/xxxx.com/root_bundle.crt"
            把上面的xxxx.com替换成你实际域名,例如我自己的是csharplike.com替换xxxx.com
        4.2.3 如果存在防火墙,则需要对外开放TCP端口443

    4.4 配置完毕记得在XAMPP控制面板里点Stop按钮,然后点Start按钮,来重启Apache令配置生效

    4.5 我们的Socket是采用先RSA后AES加密,默认设置的证书是内置的,你需要更换成自己独一无二的RSA证书
        4.5.1 生成证书
            在'Menu/Window/C#Like'打开C#Like设置面板,点"Generate RSA"按钮,会生成'Assets\C#Like\Editor\RSAPublicKey.txt'和'Assets\C#Like\Editor\RSAPrivateKey.txt'两个文件
        4.5.2 替换服务器KissServerFramework里的设置
            修改C:\KissServerFramework\KissServerFramework.json内的socketServerRSAPrivateKey成'Assets\C#Like\Editor\RSAPrivateKey.txt'的内容
        4.5.3 替换客户端C#Like里的设置
            修改'Assets\C#Like\HotUpdateScripts\Sample\SampleSocket.cs内的socketRSAPublicKey成'Assets\C#Like\Editor\RSAPublicKey.txt'的内容

5 双击运行C:/KissServerFramework/KissServerFramework.exe
    5.1 如果exe闪掉没开起了,可能是因为没有对应.NET运行时库,可以根据事件查看器的提示到微软官网下载运行时库.
    5.2 如果exe的console显示log仅为"KissFramework version : 1.0.x.x"表示端口已经被占用,可通过XAMPP的"NetStat"查看端口使用情况.

6 导出C#Like或C#LikeFree,这里选取WebGL平台
    6.1 创建一个空白的2D的Unity项目
    6.2 导入C#Like或C#LikeFree插件
    6.3 修改非热更代码的配置"Assets\C#Like\Runtime\Sample\SampleCSharpLike.cs"连接的域名和端口
        6.3.1 例如HTTP下:
            https://www.csharplike.com:10002/ReqGateway => http://127.0.0.1:9002/ReqGateway
        6.3.2 例如HTTPS下:
            https://www.csharplike.com:10002/ReqGateway => https://[Your domain]:[Your port]/ReqGateway
    6.4 在'Menu/Window/C#Like'打开C#Like设置面板,点"Rebuild Scripts"按钮,将把'Assets\C#Like\HotUpdateScripts'的脚本编译成二进制文件"Assets\StreamingAssets\output.bytes"
    6.5 可选步骤:把'Assets\C#Like\HotUpdateScripts'整个目录删掉(请自行备份哦),以验证我们的热更新脚本是真的可以热更新的!
    6.6 Unity编辑器内双击/Assets/C#Like/Scenes/SampleScene.unity打开演示场景
    6.7 在'Menu/File/Build Settings...'菜单打开'Build Settings'设置面板
        6.7.1 点击'Add Open Scenes',令'C#Like/Scenes/SampleScene'成为'Scenes In Build'里面首个场景(而非默认的空白场景'Scenes/SampleScene')
        6.7.2 选中"WebGL"后点击"Switch Platform"按钮切换当前Platform到WebGL
        6.7.3 点击"Player Setting..."按钮,然后按需修改
            6.7.3.1 "Resolution and Presentation"
                "Default Canvas Width" 600
                "Default Canvas Height" 960
                "Run In Background" "v"
            6.7.3.2 "Other Settings"
                "Api Compatibility Level" ".NET Standard 2.0"
                "Strip Engine Code" "v"
                "Managed Stripping Level" "Medium"
            6.7.3.3 "Publishing Settings"
                "Enable Exceptions" "Explicitly Thrown Exceptions Only"
                "WebAssembly Arithmetic Exceptions" "Throw"
                "Compression Format" "Gzip"
                "Data Caching" "v"
                "Decompression Fallback" "v"
        6.7.4 点击"Build"按钮,导出最终的WebGL目录
            6.7.4.1 例如C#Like导出成CSharpLikeDemo
            6.7.4.2 例如C#LikeFree导出成CSharpLikeFreeDemo

7 把C#Like或C#LikeFree导出的WebGL放入到C:\xampp\htdocs目录内
    7.1 把导出的CSharpLikeDemo目录放入C:\xampp\htdocs目录内
        7.1.1 例如我自己的配置了SSL证书的可以通过 https://www.csharplike.com/CSharpLikeDemo/index.html 来访问
        7.1.2 例如如果本地没有配置SSL证书的可以通过 http://127.0.0.1/CSharpLikeDemo/index.html 来访问
    7.2 把导出的CSharpLikeFreeDemo目录放入C:\xampp\htdocs目录内
        7.2.1 例如我自己的配置了SSL证书的可以通过 https://www.csharplike.com/CSharpLikeFreeDemo/index.html 来访问
        7.2.2 例如如果本地没有配置SSL证书的可以通过 http://127.0.0.1/CSharpLikeFreeDemo/index.html 来访问
```

便捷的KissJson

可以在热更脚本和非热更脚本中完全相同地使用,JSON字符串与对象(含热更或非热更的类或结构体)互相转换,还可以用JSONData做中介,超级方便地使用JSON

```
using CSharpLike.SubclassEx3;
using System.Collections.Generic;
using UnityEngine;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        void TestKissJson()
        {
            Debug.LogError("示范KissJson:你不能在C#Like免费版里使用@来定义JSON字符串和位运算和使用可空类型及对应运算和操作 (在完整版已支持).强烈推荐升级到完整版: " +
            "https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // @""关键字: 直接使用""代替.
            // 位运算: 使用函数来代替.
            // 可空类型: 不要使用.

            //内置数据类型包含以下:
            //byte/sbyte/char/bool/short/ushort/int/uint/long/ulong/double/float/string,
            //byte?/sbyte?/char?/bool?/short?/ushort?/int?/uint?/long?/ulong?/double?/float?,
            //List<内置数据类型>,
            //Dictionary<string, 内置数据类型>

            //内置数据类型和JSONData可以直接互相赋值和转换.

            //示范 内置数据类型转换成JSONData
            JSONData iData = 2;
            Debug.Log("JSONData iData = 2;  test iData = " + iData);//输出 2
            JSONData fData = 88.8f;
            Debug.Log("JSONData fData = 88.8f;  test fData = " + fData);//输出 88.8
```

```
            List<string> listValue = new List<string>();
            listValue.Add("test list str1");
            listValue.Add("test list str2");
            JSONData listData = listValue;
            Debug.Log("JSONData listData = listValue;  test listData = " + listData);//输出 ["test list str1","test list str2"]
            Dictionary<string, int> dictValue = new Dictionary<string, int>();
            dictValue.Add("key1", 11);
            dictValue.Add("key2", 22);
            JSONData dictData = dictValue;
            Debug.Log("JSONData dictData = dictValue;  test dictData = " + dictData);//输出 {"key1":11,"key2":22}

            //示范 JSONData转换成内置数据类型
            int iValue = iData;
            Debug.Log("int iValue = iData;  test iValue = " + iValue);//输出 2
            float fValue = fData;
            Debug.Log("float fValue = fData;  test fValue = " + fValue);//输出 88.8
            List<string> listValue2 = listData;
            string strTemp = "";
            foreach (var str in listValue2)
                strTemp += str + ",";
            Debug.Log("List<string> listValue2 = listData;  test listValue2 = " + strTemp);//输出 test list str1,test list str2,
            Dictionary<string, int> dictValue2 = dictData;
            strTemp = "";
            foreach (var item in dictValue2)
                strTemp += "(" + item.Key + "=" + item.Value + ")";
            Debug.Log("Dictionary<string, int> dictValue2 = dictData;  test dictValue2 = " + strTemp);//输出 (key1=11)(key2=22)

            //示范 JSON字符串转换成JSONData
            string strJson = "{\"str\":\"{test str\",\"i\":11,\"j\":2.3,\"k\":[3,null,{\"m\":true}],\"l\":{\"x\":1,\"y\":\"abc\"}}";
            JSONData data = KissJson.ToJSONData(strJson);
            //以["key"]来访问JSONData内的字典数据,以[数字]来访问JSONData内的数组数据
            Debug.Log("JSON string => JSONData; test data[\"str\"] = " + data["str"]);//输出 {test str
            Debug.Log("JSON string => JSONData; test data[\"i\"] = " + data["i"]);//输出 11
            Debug.Log("JSON string => JSONData; test data[\"j\"] = " + data["j"]);//输出 2.3
            Debug.Log("JSON string => JSONData; test data[\"k\"] = " + data["k"]);//输出 [3,null,{"m":true}]
            Debug.Log("JSON string => JSONData; test data[\"l\"] = " + data["l"]);//输出 {"x":1,"y":"abc"}
            Debug.Log("JSON string => JSONData; test data[\"k\"][0] = " + data["k"][0]);//输出 3
            Debug.Log("JSON string => JSONData; test data[\"l\"][\"y\"] = " + data["l"]["y"]);//输出 abc
            Debug.Log("JSON string => JSONData; test data[\"k\"][2][\"m\"] = " + data["k"][2]["m"]);//输出 true

            //示范 JSONData和内置数据类型之间直接运算
            JSONData exprData1 = 2;
            JSONData exprData2 = 3;
            JSONData exprData3 = exprData1 * exprData2;//示范加减乘除
            Debug.Log("test Math Expression;  exprData3 = exprData1 * exprData2; exprData3 = " + exprData3);//输出 6
            exprData3 = exprData1 - exprData2;
            Debug.Log("test Math Expression;  exprData3 = exprData1 - exprData2; exprData3 = " + exprData3);//输出 -1
            exprData3 *= exprData2;//exprData3=-1;exprData2=3
            Debug.Log("test Math Expression;  exprData3 *= exprData2; exprData3 = " + exprData3);//输出 -3

            iData = 2;
            if (iData > 1)//示范比较运算
                Debug.Log("test Math Expression;  iData = 2, enter if (iData > 1)");//输出
            else
                Debug.Log("test Math Expression;  iData = 2, not enter if (iData > 1)");

            //示范 JSONData转换成JSON字符串
            JSONData listData3 = JSONData.NewDictionary();//创建一个JSONData对象
            listData3.Add("key1", 10);          //可以像Dictionary的'Add(key,value)'一样添加数据
            listData3["key2"] = "test string"; //可以像Dictionary的'this[]'一样添加数据
            listData3["key3"] = JSONData.NewList(); //加一个数组(List)
            if (listData3.ContainsKey("key3"))  //可以先判断一下是否存在指定键值(这里仅示范,实际我们无需判断)
                listData3["key3"].Add(1);       //然后再往List添加数据
            listData3["key3"].Insert(0,"string2"); //往数组里指定位置插入数据
            listData3["key4"] = JSONData.NewDictionary(); //加一个子JSONData对象
            listData3["key4"]["x"] = 1;
            listData3["key4"]["y"] = 2;
            listData3["key4"]["z"] = 3;
            Debug.Log("test JSONData => JSON string; strJson = " + listData3.ToJson());//输出 {"key1":10,"key2":"test string","key3":["string2",1],"key4":
{"x":1,"y":2,"z":3}}

            //示范 游历JSONData里的数组或字典
            foreach (var item in listData3["key3"].Value as List<JSONData>)//foreach游历数组
            {
                Debug.Log("test looping through the List in JSONData using foreach; listData3[\"key3\"].Value = " + item);//输出 string2/输出 1
            }
            List<JSONData> tempList = listData3["key3"].Value as List<JSONData>;//for游历数组
            for (int i=0; i< tempList.Count; i++)
                Debug.Log("test looping through the List in JSONData using for; listData3[\"key3\"].Value = " + tempList[i]);//输出 string2/输出 1
            foreach (var item in listData3["key4"].Value as Dictionary<string, JSONData>)//foreach游历字典
            {
                Debug.Log("test looping through the Dictionary in JSONData using foreach; listData3[\"key4\"], Key=" + item.Key + ",Value=" + item.Value);//输
出 Key=x,Value=1/输出 Key=y,Value=2/输出 Key=z,Value=3
            }

            //示范 JSON字符串转换成类/结构体
            strJson = "{\"str\":\"{test str\",\"i\":11,\"j\":1,\"k\":[3,1,7],\"datas\":{\"aa\":{\"id\":1,\"name\":\"aaa\",\"v2\":{\"x\":1,\"y\":2},\"info\":
[\"a\",\"xd\",\"dt\"],\"maps\":{\"x\":1,\"y\":2}},\"bb\":{\"id\":2,\"name\":\"bbb\",\"v2\":{\"x\":3,\"y\":4},\"info\":[\"x\",\"x3d\",\"ddt\"],\"maps\":
{\"x\":2,\"y\":3}}},\"data\":{\"id\":3,\"name\":\"ccc\",\"v2\":{\"x\":3,\"y\":1},\"info\":[\"ya\",\"xyd\",\"drt\"],\"maps\":{\"x\":3,\"y\":4}}}";
            TestJsonData testJsonDdata = (TestJsonData)KissJson.ToObject(typeof(TestJsonData), strJson);//JSON字符串转换成类/结构体
            //测试输出转换后数据
            Debug.Log(testJsonDdata.str);//输出 Null
            Debug.Log(testJsonDdata.i);//输出 11
            //"j":"Monday" 或 "j":"1" 或 "j":1 都能识别为枚举'DayOfWeek.Monday'
            //建议使用数字的方式"j":1 因为转为JSON字符串的时候是数字模式
            Debug.Log(testJsonDdata.j);//输出 Monday
            Debug.Log((int)testJsonDdata.j);//输出 1
            Debug.Log(testJsonDdata.z);//输出 2
            Debug.Log(HotUpdateManager.ConvertEnumString(typeof(TestHotUpdateEnum),testJsonDdata.z));//输出 Evening
            foreach (var item in testJsonDdata.k)
                Debug.Log(item);//输出 3/输出 1/输出 7
            foreach(var datas in testJsonDdata.datas)
            {
                Debug.Log(datas.Key);//输出 aa/输出 bb
                Debug.Log(datas.Value.v2);//输出 (1.0, 2.0)/输出 (3.0, 4.0)
            }

            //示范 类/结构体转换成JSON字符串
            strTemp = KissJson.ToJson(testJsonDdata);//类/结构体转换成JSON字符串
            Debug.Log(strTemp);//输出 {"i":11,"j":1,"k":[3,1,7],"datas":{"aa":{"id":1,"name":"aaa","v2":{"x":1,"y":2},"info":["a","xd","dt"],"maps":{"x":1,"y":2}},"bb":
{"id":2,"name":"bbb","v2":{"x":3,"y":4},"info":["x","x3d","ddt"],"maps":{"x":2,"y":3}}},"data":{"id":3,"name":"ccc","v2":{"x":3,"y":1},"info":["ya","xyd","drt"],"maps":
{"x":3,"y":4}}}
        }
    }
    namespace SubclassEx3
    {
        /// <summary>
        /// 示范JSON字符串与类的转换中的子类
        /// </summary>
        public class TestJsonDataSub
        {
            public string name;
            public Vector2 v2;//你可以直接在类里转换各种类/结构体(无论热更/非热更),例如Color/Rect/Vector3/...
            public List<string> info;//直接一个List
```

```
            public Dictionary<string, int> maps;//可以转换key类型为string的字典
        }
        /// <summary>
        /// 示范JSON字符串与类的转换,被标识为不转换的类
        /// </summary>
        [KissJsonDontSerialize]//被标识为忽略转换JSON
        public class TestJsonDataSub2
        {
            public int id;
        }
        /// <summary>
        /// 示范JSON字符串与类的转换
        /// </summary>
        public class TestJsonData
        {
            [KissJsonDontSerialize]
            public string str;//被标识为忽略转换JSON
            public int i;
            public DayOfWeek j;//示范转换非热更代码的枚举
            public List<int> k;
            public Dictionary<string, TestJsonDataSub> datas;//示范字典的类
            public TestJsonDataSub data;//示范单个类
            public TestJsonDataSub2 data2;//这个转换会被忽略,因为TestJsonDataSub2这个类被标识为KissJsonDontSerialize
        }
    }
}
```

C#LikeFree里的C#特性

类的功能
这里做一个接口IAnimal,然后做一个Mammals继承于接口IAnimal, 通过这1个接口和3个类来说明类的功能

```
using UnityEngine;
using CSharpLike.Subclass;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        /// <summary>
        /// 测试非热更代码的类.
        /// </summary>
        void TestClass()
        {
            Debug.LogError("示范类: 你不能在C#Like免费版里类继承/虚函数/析构函数/构造函数(this/base) (在完整版已支持).强烈推荐升级到完整版: " +
            "https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // 析构函数/构造函数(this/base)/虚函数: 使用普通函数手动调用
            // 类继承/虚函数: 不使用,就像C语言甚至没有类的概念也一样可以做任何事,更何况你可使用类和继承接口.
            // 导致的后果是你的代码没有那么面向对象,编码风格变成面向过程.
            //测试类
            Mammals cow = new Mammals(4, 4, "cow");
            Debug.Log("cow:" + cow.GetInfo());
            //测试接口
            IAnimal bull = new Mammals(0, 4, "bull");
            bull.female = false;
            Debug.Log("bull:female=" + bull.female +
            ", CanUseTool=" + bull.CanUseTool());
        }
    }
}
```

```
using UnityEngine;

namespace CSharpLike
{
    /// <summary>
    /// 测试类
    /// </summary>
    namespace Subclass
    {
        //接口
        public interface IAnimal
        {
            int feet { get; set; }
            bool female { get; set; }
            bool CanUseTool();
        }
        //哺乳动物
        public class Mammals : IAnimal
        {
            public int breasts;
            public string name;

            public int feet { get; set; }
            public bool female { get; set; }

            public Mammals(int breasts, int feet, string name)
            {
                Debug.Log("Mammals(" + breasts + "," + feet + "," + name + ")");
                this.breasts = breasts;
            }

            public string GetInfo()
            {
                return "Mammals:" + name + " with " + feet + " feet and " + breasts + " breasts";
            }
            public bool CanUseTool()
            {
                return false;
            }
        }
    }
    /// <summary>
    /// 这个命名空间是用于测试和Subclass区分的
    /// </summary>
    namespace SubclassEx
    {
        /// <summary>
        /// 测试CSharpLike.Subclass.Mammals同类名不同命名空间用
        /// </summary>
        public class Mammals
        {
            public int breasts = 2;
            public int eyes = 2;
            public string TestNameSpace()
            {
```

```
                return "Mammals:breasts:" + breasts + ", eyes:" + eyes;
            }
        }
        public class Toys
        {
            public string name;
            public Toys(string name)
            {
                this.name = name;
            }
        }
    }
    /// <summary>
    /// 测试调用完整类名(含命名空间)方式调用热更代码
    /// </summary>
    namespace SubclassEx2
    {
        public class TestNamespace
        {
            public static string GetTestString()
            {
                return "Test string for namespace";
            }
        }
    }
}
```

委托和Lambda

测试界面有3个按钮'Test Delegate','Test Lambda','Test Bind'和一个Text组件

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        void TestDelegateAndLambda()
        {
            Debug.LogError("Test delegate and lambda:");
            //测试委托,这里通过代码方式指定按钮'Test Delegate'的按钮响应函数为OnClickDelegate
            HotUpdateManager.AddEventTrigger(GetGameObject("TestDelegate"), EventTriggerType.PointerClick, OnClickDelegate);
            //测试Lambda,这里指定按钮'Test Lambda'的按钮响应为下面Lambda代码
            HotUpdateManager.AddEventTrigger(GetGameObject("TestLambda"), EventTriggerType.PointerClick,
                (BaseEventData eventData) =>
                {
                    GetComponent<Text>("TestMessage").text = "OnClickLambda";
                    Debug.Log("On click lambda :" + eventData);
                });
        }
        /// <summary>
        /// 委托方式绑定的函数.
        /// 这里是按钮'Test Delegate'绑定按钮点击响应函数
        /// </summary>
        void OnClickDelegate(BaseEventData eventData)
        {
            GetComponent<Text>("TestMessage").text = "OnClickDelegate";
            Debug.Log("OnClickDelegate:" + eventData);
        }
        /// <summary>
        /// 由预制体里面的按钮组件直接绑定的函数.
        /// 这里是按钮'Test Bind'绑定按钮点击响应函数
        /// 这个是Button组件最简洁的点击响应函数
        /// </summary>
        void OnClickBindButton()
        {
            GetComponent<Text>("TestMessage").text = "OnClickBindButton";
            Debug.Log("OnClickBindButton:");
        }
        /// <summary>
        /// 由预制体里面的EventTrigger组件直接绑定的函数.
        /// 这里是按钮'Test Bind'绑定按钮鼠标进入响应函数
        /// 不推荐这种方式,因为同一个热更代码只有1种同类型的响应函数,且函数名字不能修改
        /// 推荐使用HotUpdateManager.AddEventTrigger的方式
        /// </summary>
        void OnPointerEnter(BaseEventData eventData)
        {
            GetComponent<Text>("TestMessage").text = "OnPointerEnter";
            Debug.Log("OnPointerEnter:" + eventData);
        }
    }
}
```

运算表达式

```
using UnityEngine;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        /// <summary>
        /// 测试运算表达式:
        /// + - * / % += -= *= /= %= > >= < <= != == && || ! ++ -- is as ?:
        /// </summary>
        void TestMathExpression()
        {
            Debug.LogError("示范运算表达式:你不能在C#Like免费版里使用位运算和可空运算.(在完整版已支持).强烈推荐升级到完整版: " +
            "https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // 位运算: 使用函数来代替.
            // 可空运算: 不使用

            int i = 1;
            int j = 2;
            int k = 100;
            //测试+ - * / %
            Debug.Log("i = 1, j = 2, test i + j = " + (i + j));//输出 3
            Debug.Log("i = 1, j = 2, test i - j = " + (i - j));//输出 -1
            Debug.Log("i = 1, j = 2, test i * j = " + (i * j));//输出 2
            Debug.Log("i = 1, j = 2, test i / j = " + (i / j));//输出 0
            Debug.Log("i = 1, j = 2, test i % j = " + (i % j));//输出 1

            //测试+= -= *= /= %=
            k += j;
            Debug.Log("j = 2, k = 100, test k += j then k = " + k);//输出 102
```

```
            k = 100;
            k -= j;
            Debug.Log("j = 2, k = 100, test k -= j then k = " + k);//输出 98
            k = 100;
            k *= j;
            Debug.Log("j = 2, k = 100, test k *= j then k = " + k);//输出 200
            k = 100;
            k /= j;
            Debug.Log("j = 2, k = 100, test k /= j then k = " + k);//输出 50
            k = 100;
            k %= j;
            Debug.Log("j = 2, k = 100, k %= j then k = " + k);//输出 0

            //测试> >= < <= != ==
            Debug.Log("i = 1, j = 2, test (i < j) = " + (i < j));//输出 True
            Debug.Log("i = 1, j = 2, test (i <= j) = " + (i <= j));//输出 True
            Debug.Log("i = 1, j = 2, test (i == j) = " + (i == j));//输出 False
            Debug.Log("i = 1, j = 2, test (i != j) = " + (i != j));//输出 True
            Debug.Log("i = 1, j = 2, test (i > j) = " + (i > j));//输出 False
            Debug.Log("i = 1, j = 2, test (i >= j) = " + (i >= j));//输出 False

            //测试&& || !
            bool b1 = true;
            bool b2 = false;
            Debug.Log("b1 = true, b2 = false, test (b1 && b2) = " + (b1 && b2));//输出 False
            Debug.Log("b1 = true, b2 = false, test (b1 || b2) = " + (b1 || b2));//输出 True
            Debug.Log("b1 = true, test (!b1) = " + (!b1));//输出 False

            //测试++ --(含前置后置)
            k = 100;
            Debug.Log("k = 100, test (k++) = " + (k++));//输出 100
            Debug.Log("finally k = " + k);////输出 101
            k = 100;
            Debug.Log("k = 100, test (++k) = " + (++k));//输出 101
            Debug.Log("finally k = " + k);////输出 101
            k = 100;
            Debug.Log("k = 100, test (k--) = " + (k--));//输出 100
            Debug.Log("finally k = " + k);////输出 99
            k = 100;
            Debug.Log("k = 100, test (--k) = " + (--k));//输出 99
            Debug.Log("finally k = " + k);////输出 99

            // 我们暂不支持 '++' '--'和运算符'[]'同时使用
            // 例如List或Dictionary或JSONData.
            //// List<int> lists = new List<int>();
            //// lists.Add(1);
            //// lists[0]++;//编译时错误,建议改成'lists[0] += 1;'
            //// ++lists[0];//运行时报错,建议改成'lists[0] += 1;'

            //测试三元表达式 ?:
            Debug.Log("i = 1, j = 2, test ((i < j) ? i : j) = " + ((i < j) ? i : j));//输出 1

            //测试转换符号is as
            object o = i;
            Debug.Log("int i = 1,object o = i, test (o is int) = " + (o is string));//输出 False
            Debug.Log("int i = 1,object o = i, test (o as string) = " + (o as string));//输出 null
        }
    }
}
```

循环语法

```
using System;
using System.Collections.Generic;
using UnityEngine;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        /// <summary>
        /// 示范循环语句
        /// for foreach continue break if-else return while do-while switch-case-default.
        /// </summary>
        void TestLoop()
        {
            Debug.LogError("示范循环语句:你不能在C#Like免费版里使用协程switch-case-default.(在完整版已支持).强烈推荐升级到完整版: " +
            "https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // switch-case-default: 直接使用"if-else"语句代替 (分支较多情况下看起来难看点和效率稍微低点).

            int x = 1;
            int y = 3;
            int z = 100;

            //示范 if-else
            if (x < y)
                Debug.Log("test 'if-else': enter 'if'");
            else if (x < z)
                Debug.Log("test 'if-else': enter 'else if'");
            else
                Debug.Log("test 'if-else': enter 'else'");

            List<Vector3> lists = new List<Vector3>();// this generic type should be AOT
                lists.Add(Vector3.zero);
                lists.Add(Vector3.one);

            //示范 for
            for (int i = 0; i<lists.Count; i++)
            {
                if (i == 2)
                {
                    Debug.Log("test 'continue':");
                    continue;
                }
                Debug.Log("test 'for': lists[" + i + "] = " + lists[i]);
            }

            //示范 foreach
            foreach(var item in lists)
            {
                Debug.Log("test 'foreach': item = " + item);
                if (item == 2)
                {
                    Debug.Log("test 'break':");
                    break;
                }
            }

            //示范 while
            whilewhile(x < y)
```

```
            {
                x++;
                Debug.Log("test 'while': x = " + x + ", y = " + y);
            }

            //示范 do-while
            x = 1;
            do
            {
                x++;
                Debug.Log("test 'do-while': x = " + x + ", y = " + y);
            } whilewhile (x < y) ;

            //示范 switch-case-default(模仿)
            //免费版下:
            if (x == 0)
                Debug.Log("test 'switch': enter 0");
            else if (x == 1)
                Debug.Log("test 'switch': enter 1");
            else if (x == 2 || x == 3)
                Debug.Log("test 'switch': enter 2 or 3");
            else
                Debug.Log("test 'switch': enter default");
            //完整版下:
            //switch(x)
            //{
            //    case 0:
            //        Debug.Log("test 'switch': enter 0");
            //        break;
            //    case 1:
            //        Debug.Log("test 'switch': enter 1");
            //        break;
            //    case 2:
            //    case 3:
            //        Debug.Log("test 'switch': enter 2 or 3");
            //        break;
            //    default:
            //        Debug.Log("test 'switch': enter default");
            //        break;
            //}
        }
    }
}
```

自定义get/set访问器

```
using UnityEngine;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        void TestGetSetAccessor()
        {
            Debug.LogError("示范get/set访问器: 你不能在C#Like免费版里使用自定义get/set访问器. (在完整版已支持).强烈推荐升级到完整版: " +
            "https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // 自定义get/set访问器: 直接使用函数来代替

            //测试自动实现的get/set访问器
            Debug.Log("before set value testGetSetAutoImp = " + testGetSetAutoImp);//输出 False
            testGetSetAutoImp = true;
            Debug.Log("after set value: testGetSetAutoImp = " + testGetSetAutoImp);//输出 True

        }

        public bool testGetSetAutoImp { get;set; }
    }
}
```

多线程

```
using System;
using System.Collections.Generic;
using System.Threading;
using UnityEngine;
using UnityEngine.UI;
using Random = UnityEngine.Random;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        void TestThread()
        {
            Debug.LogError("示范协程: 你不能在C#Like免费版里使用'lock'语句(在完整版已支持).强烈推荐升级到完整版: https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // 'lock'语句: 在非热更代码里使用,或者不用(可能引起多线程安全问题)

            //最易用的调起无参数的多线程函数方式
            HotUpdateManager.CreateThread(TestThreadRunLoop);

            //测试Task.Run
            Task.Run(() =>
            {
                Debug.LogError("Task.Run as lambda start " + DateTime.Now);
                Thread.Sleep(2000);
                Debug.LogError("Task.Run as lambda end " + DateTime.Now);
            });
            Task.Run(TestTaskRun);
        }
        void TestTaskRun()
        {
            Debug.LogError("Task.Run as delegate start " + DateTime.Now);
            Thread.Sleep(4000);
            Debug.LogError("Task.Run as delegate end " + DateTime.Now);
        }
        /// <summary>
        /// 预制体里绑定的按钮响应函数
        /// </summary>
        void OnClickTestThread()
        {
            GetComponent<Text>("TestMessage").text = "OnClickTestThread";
            //最易用的调起带参数的多线程函数方式
            JSONData jsonData = JSONData.NewDictionary();
            jsonData["id"] = Random.Range(1, 1000);
            jsonData["data"] = JSONData.NewList();
            jsonData["data"].Add("dump1");
```

```
            jsonData["data"].Add("dump2");
            List<int> testParams = new List<int>();//传入List<int>类型的参数
            testParams.Add(123);
            jsonData.SetObjectExtern("TestExternMsg", testParams);//Set extern params to thread
            HotUpdateManager.CreateThread(DoSomeWorkInThread,//这个是带参数的函数名
                jsonData,//这个是传递给函数的参数
                behaviour,//这个是当前HotUpdateBehaviour组件,如果无需回调给unity主线程处理则无需传入
                OnDoSomeWorkInThreadDone);//这个是回调给unity主线程处理的函数名,如果不需要则无需传入

            //这里修改一个数值,让前面开启的多线程打印一个log
            countInThread = Random.Range(1, 1000);

        }
        bool bExist = false;
        void OnDestroy()
        {
            bExist = true;//退出时候通知前面开启的多线程退出
        }
        object objLock = new object();
        int countInThread = 0;
        volatile int countLoopThread = 0;//测试关键字 'volatile'
        /// <summary>
        /// 不带参数的多线程函数
        /// </summary>
        void TestThreadRunLoop()
        {
            Debug.LogError("TestThreadRunLoop start");
            whilewhile (!bExist)//直到组件销毁才退出
            {
                if (countInThread > 0)
                {
                    Debug.LogError("TestThreadRunLoop:countInThread=" + countInThread);
                    countInThread = 0;
                }
                if ((++countLoopThread) >= 10000)
                    countLoopThread = 0;
                Thread.Sleep(200);//休眠0.2秒
            }
            Debug.LogError("TestThreadRunLoop end");
        }
        /// <summary>
        /// 带参数的多线程函数
        /// </summary>
        void DoSomeWorkInThread(object obj)
        {
            //我们通过'JSONData'来传递参数.传回Unity主线程的计算结果也是用这个.
            //可以在里面传递数据
            JSONData jsonData = obj as JSONData;

            List<int> testParams = jsonData.GetObjectExtern("TestExternMsg") as List<int>;//test extern param
            Debug.Log("DoSomeWorkInThread testParams.Count=" + testParams.Count);
            //这里示范在多线程内做一些工作 (你不能在本线程访问Unity的组件,例如'PlayerPrefs', 和原生C#代码一样.)
            Debug.Log("DoSomeWorkInThread(" + jsonData + ") start at " + DateTime.Now);
            long count = 0;
            int id = jsonData["id"];
            for (int i = 0; i < 10000; i++)
            {
                id += jsonData.Count;//不要直接使用'jsonData["id"] += jsonData.Count;',因为JSONData在热更新代码中不是那么高效，在这循环10000次的循环内要花太多时间了
                Interlocked.Increment(ref count);
            }
            jsonData["id"] = id;
            Debug.Log("DoSomeWorkInThread(" + jsonData + ") end at " + DateTime.Now);

            //通知主线程,我干完活了
            HotUpdateManager.OnThreadDone(netObject);//标识为多线程完成了(我们会自动在里面通知主线程执行预先设定的回调函数)
        }
        /// <summary>
        /// 当多线程工作完成后在主线程刷新UI
        /// </summary>
        void OnDoSomeWorkInThreadDone(object obj)
        {
            JSONData jsonData = obj as JSONData;
            GetComponent<Text>("TestMessage").text = "OnDoSomeWorkInThreadDone:"+ jsonData["id"];//显示在多线程计算的结果
            Debug.Log("OnDoSomeWorkInThreadDone:" + jsonData["id"]);
        }
    }
}
```

区域和宏

```
using UnityEngine;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        /// <summary>
        /// 支持unity内置的宏和自定义的宏("Edit"->"Project Settings"->"Player"->"Scripting Define Symbols").
        /// 和Unity一样,除了'UNITY_EDITOR/UNITY_EDITOR_WIN/UNITY_EDITOR_OSX/UNITY_EDITOR_LINUX'
        /// (这几个宏仅在调试模式下在编辑内生效)
        /// </summary>
        void TestMacroAndRegion()
        {
            Debug.LogError("示范区域和宏: 你不能在C#Like免费版里使用区域和宏(在完整版已支持).强烈推荐升级到完整版: https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // 宏: 在非热更脚本里定义变量来代替
            // 区域: 不使用 (你的代码在编辑器里看起来没有那么整洁).

            // 示范宏 (模拟)
            // 免费版
            if (Application.platform == RuntimePlatform.WebGLPlayer)
                Debug.Log("Test macro (simulate): is UNITY_WEBGL");
            else
                Debug.Log("Test macro (simulate): is not UNITY_WEBGL");
            // 完整版
//#if UNITY_WEBGL
//            Debug.Log("Test macro: is UNITY_WEBGL");
//#else
//            Debug.Log("Test macro: is not UNITY_WEBGL");
//#endif
            Debug.LogError("Test Macro:");
        }
    }
}
```

枚举

```
using System;
```

```
using UnityEngine;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        Debug.LogError("示范枚举: 你不能在C#Like免费版里定义枚举(在完整版已支持).强烈推荐升级到完整版: https://assetstore.unity.com/packages/slug/222256");
        // 我们提供以下方案给免费用户:
        // 枚举: 直接使用数字.
        // 在免费版,你依然可以使用在非热更脚本里定义的枚举

        Debug.Log("Test use the enum of the normal script: DayOfWeek = " + DayOfWeek.Saturday);//输出 Saturday
    }
}
```

参数修饰符

```
using UnityEngine;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        /// <summary>
            /// 示范函数参数修饰符ref out in param
        /// </summary>
        void TestModifier()
        {
            Debug.LogError("示范函数参数修饰符: 你不能在C#Like免费版里使用函数参数修饰符'ref out in param'(在完整版已支持).强烈推荐升级到完整版:"+
            " https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // ref out in param : 各自做了模拟的使用方法

            // 示范ref
            Vector3 current = Vector3.zero;
            Vector3 target = Vector3.one;
            Vector3 currentVelocity = Vector3.zero;
            Debug.Log("TestModifier ref:before:current=" + current + ",target=" + target + ",currentVelocity=" + currentVelocity);

            //免费版:(需要在非热更代码里先修改)
            SampleHowToUseModifier.currentVelocity = currentVelocity;
            current = SampleHowToUseModifier.SmoothDamp(current, target, 0.5f);
            currentVelocity = SampleHowToUseModifier.currentVelocity;
            ////完整版:
            //current = Vector3.SmoothDamp(current, target, ref currentVelocity, 0.5f);

            Debug.Log("Test ref:after:current=" + current + ",target=" + target + ",currentVelocity=" + currentVelocity);

            // 示范out
            Dictionary<string, int> dicts = new Dictionary<string, int>();
            dicts.Add("test", 1);
            int iValue;
            //免费版:(需要在非热更代码里先修改)
            if (SampleHowToUseModifier.TryGetValue(dicts, "test"))
            {
                iValue = SampleHowToUseModifier.value;
                Debug.Log("TestModifier out:iValue=" + iValue);
            }
            ////完整版:
            //if (dicts.TryGetValue("test", out iValue))
            //    Debug.Log("TestModifier out:iValue=" + iValue);

            // 示范in
            iValue = 100;
            //免费版:
            SampleHowToUseModifier.TestModifierIn("test", iValue);//You can ignore 'in' keyword 'SampleHowToUseModifier.ModifierIn("test", iValue);'
            ////完整版:
            //SampleHowToUseModifier.ModifierIn("test", in iValue);//You can ignore 'in' keyword 'SampleHowToUseModifier.ModifierIn("test", iValue);'

            // 示范params
            //免费版:(需要在非热更代码里先修改)
            SampleHowToUseModifier.TestModifierParams("free version");
            SampleHowToUseModifier.TestModifierParams("free version", "test");
            SampleHowToUseModifier.TestModifierParams("free version", "test", 1);
            SampleHowToUseModifier.TestModifierParams("free version", "test", 1, 0.5f);
            ////完整版:
            //SampleHowToUseModifier.ModifierParams("free version");
            //SampleHowToUseModifier.ModifierParams("free version", "test");
            //SampleHowToUseModifier.ModifierParams("free version", "test", 1);
            //SampleHowToUseModifier.ModifierParams("free version", "test", 1, 0.5f);
        }
    }
}
```

免费版的非热更代码里对应修改:

```
using UnityEngine;

namespace CSharpLike
{
    public class SampleHowToUseModifier
    {
        #region Modifier ref
        public static Vector3 currentVelocity = Vector3.zero;
        public static Vector3 SmoothDamp(Vector3 current, Vector3 target, float smoothTime)
        {
            return Vector3.SmoothDamp(current, target, ref currentVelocity, smoothTime);
        }
        #endregion
        #region Modifier out
        public static int value = 0;
        public static bool TryGetValue(Dictionary<string, int> dics, string key)
        {
            return dics.TryGetValue(key, out value);
        }
        #endregion
        #region Modifier in
        public static void TestModifierIn(string str, int iValue)
        {
            ModifierIn(str,in iValue);
        }
        public static void ModifierIn(string str, in int iValue)
        {
            Debug.Log("ModifierIn:"+str + iValue);
        }
        #endregion
        #region Modifier params
        public static void TestModifierParams(string str)
        {
```

```
            ModifierParams(str);
        }
        public static void TestModifierParams(string str, object v1)
        {
            ModifierParams(str, v1);
        }
        public static void TestModifierParams(string str, object v1, object v2)
        {
            ModifierParams(str, v1, v2);
        }
        public static void TestModifierParams(string str, object v1, object v2, object v3)
        {
            ModifierParams(str, v1, v2, v3);
        }
        public static void ModifierParams(string str, params object[] values)
        {
            string strTemp = "ModifierParams:" + str;
            foreach (var value in values)
                strTemp += "," + value;
            Debug.Log(strTemp);
        }
        #endregion
    }
}
```

默认参数和函数重载

```
using UnityEngine;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        /// <summary>
        /// test function overloading and function param default value
        /// </summary>
        void TestOverloadingAndDefaultValue()
        {
            Debug.LogError("示范协程: 你不能在C#Like免费版里使用默认参数和函数重载 (在完整版已支持).强烈推荐升级到完整版: " +
            "https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // 函数重载 : 用不同的函数名
            // 默认参数 : 移除默认值,乖乖传所有参数
        }
    }
}
```

关键字

```
using UnityEngine;//示范using 命令
using System;
using Random = UnityEngine.Random;//示范using 别名
using System.Text;
using System.IO;

namespace CSharpLike
{
    public partial class ExampleCSharp : LikeBehaviour
    {
        /// <summary>
        /// test not classify keyword
        /// </summary>
        void TestKeyword()
        {
            Debug.LogError("示范协程: 你不能在C#Like免费版里使用'$ sizeof unsafe #pragma #warning #error'关键字 (在完整版已支持).强烈推荐升级到完整版: " +
            "https://assetstore.unity.com/packages/slug/222256");
            // 我们提供以下方案给免费用户:
            // '$""': 使用string.Format (你的代码在编辑器看起来没有那么清爽).
            // '@""': 直接使用""代替
            // sizeof and unsafe: 直接使用数字
            // #pragma #warning #error: 不使用

            //内置数据类型
            //Int32/UInt32/Int64/UInt64/Int16/UInt16/Char/Single/Double/Decimal/Boolean/SByte/Byte/String.
            //等效于
            //int/uint/long/ulong/short/ushort/char/float/double/decimal/bool/sbyte/byte/string.
            Int32 i = 123;//equal to 'int i = 123;'
            Debug.Log("test Int32 i = " + i);//输出 123
        }
    }
}
```

联系我: csharplike@qq.com