ET 框架小游戏--斗地主源码学习

deepwaterooo

January 31, 2023

Contents

1 几个项目，哪里先后是如何运行的？文件太多，傻傻分不清楚，可是一定会弄清楚的

1.1 ETModel.Init.cs: 不可热更新, 应该是它先起始，再启热更新层的启动文件

```
namespace ETModel {
    public class Init : MonoBehaviour {

        private async void Start() {
            try {
                if (!Application.unityVersion.StartsWith("2017.4")) {
                    Log.Error($" 新人请使用 Unity2017.4 版本，减少跑 demo 遇到的问题！ 下载地址:\n https:// unity3d.com/cn/unit
                }
                SynchronizationContext.SetSynchronizationContext(OneThreadSynchronizationContext.Instance); // 异步线程等的上
                DontDestroyOnLoad(gameObject);
                Game.EventSystem.Add(DLLType.Model, typeof(Init).Assembly); // Unity.Model 里面的代码不能热更新，通常将游戏中不
                Game.Scene.AddComponent<GlobalConfigComponent>(); // 读取全局配置，不知道是否会触发什么回调
                Game.Scene.AddComponent<NetOuterComponent>();     // 客户端需要与登录服，网关服通消息，必须挂这个
                Game.Scene.AddComponent<ResourcesComponent>();
                Game.Scene.AddComponent<PlayerComponent>();
                Game.Scene.AddComponent<UnitComponent>();
                Game.Scene.AddComponent<UIComponent>();
                // 斗地主客户端自定义全局组件
                // 用于保存玩家本地数据
                Game.Scene.AddComponent<ClientComponent>();
                // 下载 ab 包
                await BundleHelper.DownloadBundle();
                Game.Hotfix.LoadHotfixAssembly();
                // 加载配置
                Game.Scene.GetComponent<ResourcesComponent>().LoadBundle("config.unity3d");
                Game.Scene.AddComponent<ConfigComponent>();
                Game.Scene.GetComponent<ResourcesComponent>().UnloadBundle("config.unity3d");
                Game.Scene.AddComponent<OpcodeTypeComponent>();
                Game.Scene.AddComponent<MessageDispatherComponent>();
                Game.Hotfix.GotoHotfix();
                Game.EventSystem.Run(EventIdType.TestHotfixSubscribMonoEvent, "TestHotfixSubscribMonoEvent");
            }
            catch (Exception e) {
                Log.Error(e);
            }
        }
        private void Update() {
            OneThreadSynchronizationContext.Instance.Update();
            Game.Hotfix.Update?.Invoke();
```

```
            Game.EventSystem.Update();
        }
        private void LateUpdate() {
            Game.Hotfix.LateUpdate?.Invoke();
            Game.EventSystem.LateUpdate();
        }
        private void OnApplicationQuit() {
            Game.Hotfix.OnApplicationQuit?.Invoke();
            Game.Close();
        }
    }
}
```

1.2 ETModel.Hotfix:

```
namespace ETModel {
    public sealed class Hotfix : Object {

#if ILRuntime
        private ILRuntime.Runtime.Enviorment.AppDomain appDomain;
#else
        private Assembly assembly;
#endif
        private IStaticMethod start;
        public Action Update;
        public Action LateUpdate;
        public Action OnApplicationQuit;

        public Hotfix() {
        }

        public void GotoHotfix() {
#if ILRuntime
            ILHelper.InitILRuntime(this.appDomain); // 这个方法里作: ILRuntime 热更新的必要的加载
#endif
            // 在 ETModel.Int 中, 加载 Hotfix.dll 的程序集, 指定的开始函数为 ETHotfix.Init.Start 方法
            this.start.Run(); // <<<<<<<<<<<<<<<<<<<< 就是这些细节的地方, 跟着跟着就会跟丢, 要看下面的地方, 有定义
        }

        public List<Type> GetHotfixTypes() {
#if ILRuntime
            if (this.appDomain == null) {
                return new List<Type>();
            }
            return this.appDomain.LoadedTypes.Values.Select(x => x.ReflectionType).ToList();
#else
            if (this.assembly == null) {
                return new List<Type>();
            }
            return this.assembly.GetTypes().ToList();
#endif
        }

        public void LoadHotfixAssembly() {
            Game.Scene.GetComponent<ResourcesComponent>().LoadBundle($"code.unity3d");
#if ILRuntime
            Log.Debug($" 当前使用的是 ILRuntime 模式");
            this.appDomain = new ILRuntime.Runtime.Enviorment.AppDomain();
            GameObject code = (GameObject)Game.Scene.GetComponent<ResourcesComponent>().GetAsset("code.unity3d", "Code");
            byte[] assBytes = code.Get<TextAsset>("Hotfix.dll").bytes;
            byte[] mdbBytes = code.Get<TextAsset>("Hotfix.pdb").bytes;
            using (MemoryStream fs = new MemoryStream(assBytes))
                using (MemoryStream p = new MemoryStream(mdbBytes)) {
                    this.appDomain.LoadAssembly(fs, p, new Mono.Cecil.Pdb.PdbReaderProvider());
                }
            this.start = new ILStaticMethod(this.appDomain, "ETHotfix.Init", "Start", 0); // <<<<<<<<<< 这里说, 起始方法是这
#else
            Log.Debug($" 当前使用的是 Mono 模式");
            GameObject code = (GameObject)Game.Scene.GetComponent<ResourcesComponent>().GetAsset("code.unity3d", "Code");
            byte[] assBytes = code.Get<TextAsset>("Hotfix.dll").bytes;
            byte[] mdbBytes = code.Get<TextAsset>("Hotfix.mdb").bytes;
            this.assembly = Assembly.Load(assBytes, mdbBytes);
            Type hotfixInit = this.assembly.GetType("ETHotfix.Init");
            this.start = new MonoStaticMethod(hotfixInit, "Start");
#endif
            Game.Scene.GetComponent<ResourcesComponent>().UnloadBundle($"code.unity3d");
        }
```

```
    }
}
```

1.3 ETHotfix.Init.cs：这里就回到了热更新域里的起始配置点

```
namespace ETHotfix {
    public static class Init {

        public static void Start() {
            try {
#if ILRuntime
                if (!Define.IsILRuntime) {
                    Log.Error("mono 层是 mono 模式, 但是 Hotfix 层是 ILRuntime 模式");
                }
#else
                if (Define.IsILRuntime) {
                    Log.Error("mono 层是 ILRuntime 模式, Hotfix 层是 mono 模式");
                }
#endif
                Game.Scene.ModelScene = ETModel.Game.Scene;
                // 注册热更层回调
                ETModel.Game.Hotfix.Update = () => { Update(); };
                ETModel.Game.Hotfix.LateUpdate = () => { LateUpdate(); };
                ETModel.Game.Hotfix.OnApplicationQuit = () => { OnApplicationQuit(); };

                Game.Scene.AddComponent<UIComponent>();
                Game.Scene.AddComponent<OpcodeTypeComponent>();
                Game.Scene.AddComponent<MessageDispatherComponent>();
                // 加载热更配置
                ETModel.Game.Scene.GetComponent<ResourcesComponent>().LoadBundle("config.unity3d");
                Game.Scene.AddComponent<ConfigComponent>();
                ETModel.Game.Scene.GetComponent<ResourcesComponent>().UnloadBundle("config.unity3d");
                UnitConfig unitConfig = (UnitConfig)Game.Scene.GetComponent<ConfigComponent>().Get(typeof(UnitConfig), 1001
                Log.Debug($"config {JsonHelper.ToJson(unitConfig)}");
                // Game.EventSystem.Run(EventIdType.InitSceneStart);
                Game.EventSystem.Run(EventIdType.LandlordsInitSceneStart); // 这个特定事件是怎么运行的？不是标签系统吗，加载的时
            }
            catch (Exception e) {
                Log.Error(e);
            }
        }
        public static void Update() {
            try {
                Game.EventSystem.Update();
            }
            catch (Exception e) {
                Log.Error(e);
            }
        }
        public static void LateUpdate() {
            try {
                Game.EventSystem.LateUpdate();
            }
            catch (Exception e) {
                Log.Error(e);
            }
        }
        public static void OnApplicationQuit() {
            Game.Close();
        }
    }
}
```

1.4 ETHotfix.InitSceneStart_CreateLandlordsLogin.cs

```
namespace ETHotfix {

 // 加载的时候，扫描到的标签系统，这个标签，就对应了这么个事件
    [Event(EventIdType.LandlordsInitSceneStart)]
    public class InitSceneStart_CreateLandlordsLogin : AEvent {
        public override void Run() {
            // 创建登录界面
            UI ui = Game.Scene.GetComponent<UIComponent>().Create(UIType.LandlordsLogin);
        }
    }
}
```

2 服务器端的逻辑

- 感觉上面的客户端的起始逻辑大致还是能够理得清的
- 这么无限地看下去, 不知道到哪年哪月才能真正看得完, 还是得赶快把自己的小服务器给弄起来
- 那么就集中到自己想要实现的功能点:
  - 一个热更新资源包的文件服务器, 及下载文件的相关逻辑
  - 注册登录服:Realm
  - 网关服：Gate
  - DB 服务器：用户信息, 或更进一步的, 游戏数据数据库服务器
- 或者弄一个 AllServer 一个服务器解决一切问题，比较简单一点儿
- 因为自己的服务器不是网络多人游戏，不涉及折到点儿的 UI 以及逻辑以及热更新的折分，只处理资源包文件服，登录登出拿取用户数据背个数据库，什么 Map 地图什么的全不涉及，应该能够简单狠多，可是现在大的框架把据不透，小的知识点还没能理解透，所以过程中应该还会有些曲折波折。
- 所以现在今天晚上看服务器端的源码，试图着重把上面与自己关系更为重要的几个问题解决掉：哪些功能模块是必须的，哪些是可以砍砍杀杀砍掉的?
- 现在看服务器客户端哪一端，都还感觉是盲人摸象，不知道是怎么回事，怎么才能够把具备（ET 框架具备吗?）动态热更新能力的服务器端改造成只具备自己想要的功能的减小精悍的小小服务器（我的服务器可以不需要动态热更新能力。只需要我的客户端能够动态热更新资源包就可以了)?
- 今天晚上把服务器端的源码理一理，就该试着用 ET 框架的 framework 来改造成自已的小小的服务器，完成自已游戏的最后最具挑战的部分了。