ET 框架学习笔记 - - 自己需要这样一个总结文档来帮助总结与急速重构自己的游戏

deepwaterooo

May 14, 2023

Contents

1 客户端场景组件：客户端大致的起始过程 1
1.1 Entry.cs: 指定的起始类，会触发三类回调，公用组件类的加载，和其它 1
1.2 EntryEvent1_InitShare: 第一类，，公用组件类的加载，公用的几大组件 1
1.2.1 CurrentScenesComponent: 可以用来管理多个客户端场景，比如大世界会加载多块场景 (是说，大地图可以分 10 块 8 块小地图吗？) 1
1.2.2 CurrentScenesComponentSystem: CurrentScene() 方法，返回当前场景 . . . 2
1.2.3 ObjectWait: 也有生成系 2
1.2.4 PlayerComponent: 2
1.2.5 PlayerComponentSystem: 生成系，到处都要用它 2
1.3 AfterCreateCurrentScene_AddComponent:【UIComponent】【ResourcesLoaderComponent】 2
1.3.1 UIComponent: 管理 Scene 上的 UI 2
1.3.2 UIComponentSystem: 管理 Scene 上的 UI: 这个是组件生成管理系统，负责添加与删除。【UIEventComponent】是 UI 上的 UI 事件组件系统 3
1.3.3 ResourcesLoaderComponent: 相关的资源加载，这个文件里有生成系 3
1.4 EntryEvent2_InitServer: 前面 1 里，两端公用组件准备好了，现在就起始服务器？服务端的几大组件： 3
1.4.1 ActorMessageSenderComponent: 发送普通 actor 消息 4
1.4.2 ActorLocationSenderComponent: 发送 location actor 消息 4
1.4.3 LocationProxyComponent: 访问 location server 的组件 4
1.4.4 ActorMessageDispatcherComponent: Actor 消息分发组件 4
1.4.5 ServerSceneManagerComponent: 可以去对比，两端的管理者组件，有什么不同？ 4
1.5 EntryEvent3_InitClient: 客户端 4
1.5.1 ResourcesComponent: 热更新资源包等的处理 5
1.5.2 GlobalComponent: 不知道是干什么的，Unity 里好像是 Root 根节点下的一个节点，组件？ 5
1.6 前面三件（【公用组件】,【服务器】,【客户端】的应用程序启动完成）触发 UI 变更: 这个 UI 订阅说，一被通知，就创建注册登录界面 5

2 ClientComponent ClientScene 等客户端相关：有点儿理不清 5
2.1 ClientSceneManagerComponent: 是否，相当于，它是 SceneType 的管理者，就是先前各种服，注册登录服，网关服、匹配服等的管理者，大概主要还是地址传送 5

3 客户端场景与客户端场景加工厂 6
3.1 SceneChangeHelper: 场景切换协程 . . . 6
3.1.1 Unit: Unit 究竟是什么，干什么的？像是游戏的一个最小单位，有位置与旋转参数 6
3.1.2 UnitComponent: 组件 . . . 7
3.1.3 UnitComponentSystem: 生成系. 感觉这个系统不太懂 . . . 7
3.1.4 UnitHelper: 帮助在不同使用情境下，拿到 unit . . . 7
3.2 SceneFactory: ClientScene: 添加三组件：【CurrentScenesComponent】【PlayerComponent】【ObjectWait】。 . . . 7
3.2.1 UnitFactory: 为什么我抓出两个不一样的定义，还没弄明白 . . . 8

4 标签系: 标签系统重构了，现分为几个类型 8
4.1 ComponentOfAttribute : Attribute . . . 8
4.2 ComponentView: MonoBehaviour . . . 9
4.3 ComponentViewEditor: Editor . . . 9

5 UI 上的事件驱动系统： 9
5.1 EventType . . . 9
5.2 由 AppStartInitFinish 事件所触发的 CreateLoginUI . . . 9
5.3 由 LoginFinish 事件所触发的 CreateLobbyUI . . . 9
5.4 SceneChangeStart_AddComponent: 开始切换场景的时候，就自动添加【OperaComponent】组件。现在对场景这块儿还不够熟悉 . . . 10

6 Helper 类的总结：【但凡点击回调方法，就变成 Helper 类！】为什么就变成了这么一个个的帮助类呢？ 10
6.1 LoginHelper.cs . . . 10
6.2 EnterRoomHelper.cs . . . 10
6.3 UIHelper.cs: 负责 UI 界面上的组件的，添加与删除，异步完成 . . . 11
6.4 SceneChangeHelper: 场景切换协程 . . . 12

7 UI 控件的生产事件机制流程：以前的专用工厂再包装为 UI 上的事件机制 12
7.1 LoginHelper 发布 EventType.LoginFinish() 事件 . . . 12
7.2 LoginFinish_RemoveLoginUI: 一般对应两个事件，旧视图的去除，与新视图的添加 . 13
7.3 LoginFinish_CreateLobbyUI: 创建新视图 . . . 13
7.4 UIHelper: 帮助类，来添加或是移除 UI 上的可装可折的组件 . . . 13
7.5 UIComponentSystem: 管理 Scene 上的 UI: 这个是组件生成管理系统，负责添加与删除。【UIEventComponent】是 UI 上的 UI 事件组件系统 . . . 13
7.6 UIEventComponentSystem: 管理所有 UI GameObject 以及 UI 事件: 应该主要是 UI 控件相关的事件。【自顶向下】的组件系统 . . . 13
7.7 AUIEvent: 跟下面的 UIEventComponent 关系是？ . . . 14
7.8 UIEventComponent: 管理所有 UI GameObject . . . 14
7.9 UIEventComponentSystem: 生成系，管理所有 UI GameObject 以及 UI 事件: 应该主要是 UI 控件相关的事件。【自顶向下】的组件系统 . . . 14
7.10 UILoginEvent: 一个实体类的例子，具体的工厂生产逻辑 . . . 15
7.11 UILobbyEvent: 再加一个实体类的例子 . . . 15

8 Session 相关：进行间通信 16
8.1 SessionComponent . . . 16
8.2 SessionComponentDestroySystem: 【销毁系】：只负责用户掉线，或是下线后的自动移除会话框 . . . 16
8.3 OperaComponentSystem: 一个拿会话框必消息的使用场景 . . . 16

9 Player: 玩家相关，添加的地方，以及使用【这里还是有点儿糊涂】 17
9.1 Player: 玩家 17
9.2 PlayerComponent: 17
9.3 服务器端的 PlayerComponent 17
9.4 服务器端 PlayerComponentSystem 17
9.5 服务器端 SceneFactory-CreateServerScene 时【网关服】会添加【PlayerComponent】玩家组件 17
9.6 SessionPlayerComponentSystem 18
9.7 SessionPlayerComponent: 会话框里，会保留客户端玩家 PlayerId 18

10 ResourcesComponent 资源包管理器相关：有时候，拖拉机游戏里会需要拿它来加载图片 18
10.1 ResourcesComponent: 同文件有其生成系 18
10.2 客户端 ConfigLoader 的 Invoke 标签下：在根控件 Root 下添加资源管理器组件 . . . 19

11 整个框架：ET 7.2 ＋ YooAssets ＋ luban ＋ FairGUI 19

12 写在最后：反而是自己每天查看一再更新的 20

13 现在的修改内容，记忆 20

14 TODO 今天晚上把几个消息抓全了，免得一堆的报错 21

1 客户端场景组件：客户端大致的起始过程

1.1 Entry.cs: 指定的起始类，会触发三类回调，公用组件类的加载，和其它

```
public static class Entry {
    public static void Init() {
    }
    public static void Start() {
        StartAsync().Coroutine();
    }
    // 【各种应用程序，第三方库等的初始化 】
    private static async ETTask StartAsync() {
        WinPeriod.Init();

        MongoHelper.Init();
        ProtobufHelper.Init();

        Game.AddSingleton<NetServices>();
        Game.AddSingleton<Root>();
        await Game.AddSingleton<ConfigComponent>().LoadAsync();

        // 不知道：加这三个是在做什么？它没有起有意义的名字，但总之，它是事件，会触发相应的回调
        await EventSystem.Instance.PublishAsync(Root.Instance.Scene, new EventType.EntryEvent1());
        await EventSystem.Instance.PublishAsync(Root.Instance.Scene, new EventType.EntryEvent2());
        await EventSystem.Instance.PublishAsync(Root.Instance.Scene, new EventType.EntryEvent3());
    }
}
```

1.2 EntryEvent1_InitShare: 第一类，，公用组件类的加载，公用的几大组件

```
// 公用的相关组件的初始化：
[Event(SceneType.Process)]
public class EntryEvent1_InitShare: AEvent<EventType.EntryEvent1> {
    protected override async ETTask Run(Scene scene, EventType.EntryEvent1 args) {
        Root.Instance.Scene.AddComponent<NetThreadComponent>();
        Root.Instance.Scene.AddComponent<OpcodeTypeComponent>();
        Root.Instance.Scene.AddComponent<MessageDispatcherComponent>();
        Root.Instance.Scene.AddComponent<NumericWatcherComponent>();
        Root.Instance.Scene.AddComponent<AIDispatcherComponent>();
        Root.Instance.Scene.AddComponent<ClientSceneManagerComponent>();
        await ETTask.CompletedTask;
    }
}
```

1.2.1 CurrentScenesComponent: 可以用来管理多个客户端场景，比如大世界会加载多块场景 (是说，大地图可以分 10 块 8 块小地图吗？)

```
// 可以用来管理多个客户端场景，比如大世界会加载多块场景 (意思是说，大地图可以分 10 块 8 块小地图吗？ )
[ComponentOf(typeof(Scene))]
public class CurrentScenesComponent: Entity, IAwake {
    public Scene Scene { get; set; }
}
```

1.2.2 CurrentScenesComponentSystem: CurrentScene() 方法，返回当前场景

```
public static class CurrentScenesComponentSystem {
    public static Scene CurrentScene(this Scene clientScene) {
        return clientScene.GetComponent<CurrentScenesComponent>()?.Scene;
    }
}
```

1.2.3 ObjectWait: 也有生成系

```
[ComponentOf]
public class ObjectWait: Entity, IAwake, IDestroy {
    public Dictionary<Type, object> tcss = new Dictionary<Type, object>();
}
```

1.2.4 PlayerComponent:

```
[ComponentOf(typeof(Scene))]
public class PlayerComponent: Entity, IAwake {
    public long MyId { get; set; }
}
```

1.2.5 PlayerComponentSystem: 生成系，到处都要用它

```
[FriendOf(typeof(PlayerComponent))]
public static class PlayerComponentSystem {
    public static void Add(this PlayerComponent self, Player player) {
        self.idPlayers.Add(player.Id, player);
    }
    public static Player Get(this PlayerComponent self, long id) {
        self.idPlayers.TryGetValue(id, out Player gamer);
        return gamer;
    }
    public static void Remove(this PlayerComponent self, long id) {
        self.idPlayers.Remove(id);
    }
    public static Player[] GetAll(this PlayerComponent self) {
        return self.idPlayers.Values.ToArray();
    }
}
```

1.3 AfterCreateCurrentScene_AddComponent:【UIComponent】【ResourcesLoaderComponent】

```
[Event(SceneType.Current)]
public class AfterCreateCurrentScene_AddComponent: AEvent<EventType.AfterCreateCurrentScene> {
    protected override async ETTask Run(Scene scene, EventType.AfterCreateCurrentScene args) {
        scene.AddComponent<UIComponent>();
        scene.AddComponent<ResourcesLoaderComponent>();
        await ETTask.CompletedTask;
    }
}
```

1.3.1 UIComponent: 管理 Scene 上的 UI

```
// 管理 Scene 上的 UI
[ComponentOf(typeof(Scene))]
public class UIComponent: Entity, IAwake {
    public Dictionary<string, UI> UIs = new Dictionary<string, UI>();
}
```

1.3.2 UIComponentSystem: 管理 Scene 上的 UI: 这个是组件生成管理系统，负责添加与删除。【UIEventComponent】是 UI 上的 UI 事件组件系统

```
// 管理 Scene 上的 UI: 这个是组件生成管理系统，负责添加与删除。【UIEventComponent】是 UI 上的 UI 事件组件系统
[FriendOf(typeof(UIComponent))]
public static class UIComponentSystem {
    public static async ETTask<UI> Create(this UIComponent self, string uiType, UILayer uiLayer) {
        UI ui = await UIEventComponent.Instance.OnCreate(self, uiType, uiLayer);
        self.UIs.Add(uiType, ui);
        return ui;
    }
    public static void Remove(this UIComponent self, string uiType) {
        if (!self.UIs.TryGetValue(uiType, out UI ui)) {
            return;
        }
        UIEventComponent.Instance.OnRemove(self, uiType);

        self.UIs.Remove(uiType);
        ui.Dispose();
    }
    public static UI Get(this UIComponent self, string name) {
        UI ui = null;
        self.UIs.TryGetValue(name, out ui);
        return ui;
    }
}
```

1.3.3 ResourcesLoaderComponent: 相关的资源加载，这个文件里有生成系

```
[ComponentOf(typeof(Scene))]
public class ResourcesLoaderComponent: Entity, IAwake, IDestroy {
    public HashSet<string> LoadedResource = new HashSet<string>();
}
```

1.4 EntryEvent2_InitServer: 前面 1 里，两端公用组件准备好了，现在就起始服务器？服务端的几大组件:

```
[Event(SceneType.Process)]
public class EntryEvent2_InitServer: AEvent<ET.EventType.EntryEvent2> {
    protected override async ETTask Run(Scene scene, ET.EventType.EntryEvent2 args) {
        // 发送普通 actor 消息
        Root.Instance.Scene.AddComponent<ActorMessageSenderComponent>();
        // 发送 location actor 消息
        Root.Instance.Scene.AddComponent<ActorLocationSenderComponent>();
        // 访问 location server 的组件
        Root.Instance.Scene.AddComponent<LocationProxyComponent>();
        Root.Instance.Scene.AddComponent<ActorMessageDispatcherComponent>();
        Root.Instance.Scene.AddComponent<ServerSceneManagerComponent>();
        Root.Instance.Scene.AddComponent<RobotCaseComponent>();
        Root.Instance.Scene.AddComponent<NavmeshComponent>();
        StartProcessConfig processConfig = StartProcessConfigCategory.Instance.Get(Options.Instance.Process);
        switch (Options.Instance.AppType) {
        case AppType.Server: {
            Root.Instance.Scene.AddComponent<NetInnerComponent, IPEndPoint>(processConfig.InnerIPPort);
            var processScenes = StartSceneConfigCategory.Instance.GetByProcess(Options.Instance.Process);
            foreach (StartSceneConfig startConfig in processScenes) {
                await SceneFactory.CreateServerScene(ServerSceneManagerComponent.Instance, startConfig.Id, startConfig.Inst
                                                     startConfig.Type, startConfig);
            }
            break;
        }
        case AppType.Watcher: {
            StartMachineConfig startMachineConfig = WatcherHelper.GetThisMachineConfig();
            WatcherComponent watcherComponent = Root.Instance.Scene.AddComponent<WatcherComponent>();
            watcherComponent.Start(Options.Instance.CreateScenes);
            Root.Instance.Scene.AddComponent<NetInnerComponent, IPEndPoint>(NetworkHelper.ToIPEndPoint($"{startMachineConfi
            break;
        }
        case AppType.GameTool:
            break;
        }
        if (Options.Instance.Console == 1) {
            Root.Instance.Scene.AddComponent<ConsoleComponent>();
        }
    }
}
```

1.4.1 ActorMessageSenderComponent: 发送普通 actor 消息

```
[ComponentOf(typeof(Scene))]
public class ActorMessageSenderComponent: Entity, IAwake, IDestroy {
    public const long TIMEOUT_TIME = 40 * 1000;
    public static ActorMessageSenderComponent Instance { get; set; }
    public int RpcId;
    public readonly SortedDictionary<int, ActorMessageSender> requestCallback = new SortedDictionary<int, ActorMessageSende
    public long TimeoutCheckTimer;
    public List<int> TimeoutActorMessageSenders = new List<int>();
}
```

1.4.2 ActorLocationSenderComponent: 发送 location actor 消息

```
[ComponentOf(typeof(Scene))]
public class ActorLocationSenderComponent: Entity, IAwake, IDestroy {
    public const long TIMEOUT_TIME = 60 * 1000;
    public static ActorLocationSenderComponent Instance { get; set; }
    public long CheckTimer;
}
```

1.4.3 LocationProxyComponent: 访问 location server 的组件

```
[ComponentOf(typeof(Scene))]
public class LocationProxyComponent: Entity, IAwake, IDestroy {
    [StaticField]
    public static LocationProxyComponent Instance;
}
```

1.4.4 ActorMessageDispatcherComponent: Actor 消息分发组件

```
public class ActorMessageDispatcherInfo {
    public SceneType SceneType { get; }
    public IMActorHandler IMActorHandler { get; }
    public ActorMessageDispatcherInfo(SceneType sceneType, IMActorHandler imActorHandler) {
        this.SceneType = sceneType;
        this.IMActorHandler = imActorHandler;
    }
}
// Actor 消息分发组件
[ComponentOf(typeof(Scene))]
public class ActorMessageDispatcherComponent: Entity, IAwake, IDestroy, ILoad {
    [StaticField]
    public static ActorMessageDispatcherComponent Instance;
    public readonly Dictionary<Type, List<ActorMessageDispatcherInfo>> ActorMessageHandlers = new();
}
```

1.4.5 ServerSceneManagerComponent: 可以去对比，两端的管理者组件，有什么不同？

```
[ComponentOf(typeof(Scene))]
public class ServerSceneManagerComponent: Entity, IAwake, IDestroy {
    [StaticField]
    public static ServerSceneManagerComponent Instance;
}
```

1.5 EntryEvent3_InitClient: 客户端

```
[Event(SceneType.Process)]
public class EntryEvent3_InitClient: AEvent<ET.EventType.EntryEvent3> {
    protected override async ETTask Run(Scene scene, ET.EventType.EntryEvent3 args) {
        // 加载配置
        Root.Instance.Scene.AddComponent<ResourcesComponent>();

        Root.Instance.Scene.AddComponent<GlobalComponent>();
        await ResourcesComponent.Instance.LoadBundleAsync("unit.unity3d");

        Scene clientScene = await SceneFactory.CreateClientScene(1, "Game");
        await EventSystem.Instance.PublishAsync(clientScene, new EventType.AppStartInitFinish()); // 应用程序启动结束
    }
}
```

1.5.1 ResourcesComponent: 热更新资源包等的处理

```
[ComponentOf]
public class ResourcesComponent: Entity, IAwake, IDestroy {
    public static ResourcesComponent Instance { get; set; }
    public AssetBundleManifest AssetBundleManifestObject { get; set; }
    public Dictionary<int, string> IntToStringDict = new Dictionary<int, string>();
    public Dictionary<string, string> StringToABDict = new Dictionary<string, string>();
    public Dictionary<string, string> BundleNameToLowerDict = new Dictionary<string, string>() { { "StreamingAssets", "Stre
    public readonly Dictionary<string, Dictionary<string, UnityEngine.Object>> resourceCache =
        new Dictionary<string, Dictionary<string, UnityEngine.Object>>();
    public readonly Dictionary<string, ABInfo> bundles = new Dictionary<string, ABInfo>();

    // 缓存包依赖，不用每次计算
    public readonly Dictionary<string, string[]> DependenciesCache = new Dictionary<string, string[]>();
}
```

1.5.2 GlobalComponent: 不知道是干什么的，Unity 里好像是 Root 根节点下的一个节点，组件？

```
[ComponentOf(typeof(Scene))]
public class GlobalComponent: Entity, IAwake {
    [StaticField]
    public static GlobalComponent Instance;
    public Transform Global;
    public Transform Unit { get; set; }
    public Transform UI;
}
```

1.6 前面三件（【公用组件】,【服务器】,【客户端】的应用程序启动完成）触发 UI 变更: 这个 UI 订阅说，一被通知，就创建注册登录界面

```
[Event(SceneType.Client)]
public class AppStartInitFinish_CreateLoginUI: AEvent<EventType.AppStartInitFinish> {
    protected override async ETTask Run(Scene scene, EventType.AppStartInitFinish args) {
        await UIHelper.Create(scene, UIType.UILogin, UILayer.Mid);
    }
}
```

- 感觉接下来就是相对熟悉的程序。再跟就去跟不熟悉的其它细节程序

2 ClientComponent ClientScene 等客户端相关：有点儿理不清

2.1 ClientSceneManagerComponent: 是否，相当于，它是 SceneType 的管理者，就是先前各种服，注册登录服，网关服、匹配服等的管理者，大概主要还是地址传送

```
[ComponentOf(typeof(Scene))]
public class ClientSceneManagerComponent: Entity, IAwake, IDestroy {
    [StaticField]
    public static ClientSceneManagerComponent Instance;
}
```

3 客户端场景与客户端场景加工厂

3.1 SceneChangeHelper: 场景切换协程

```
public static class SceneChangeHelper {
    // 场景切换协程
    public static async ETTask SceneChangeTo(Scene clientScene, string sceneName, long sceneInstanceId) {
        clientScene.RemoveComponent<AIComponent>();

        CurrentScenesComponent currentScenesComponent = clientScene.GetComponent<CurrentScenesComponent>();
        currentScenesComponent.Scene?.Dispose(); // 删除之前的 CurrentScene, 创建新的
        Scene currentScene = SceneFactory.CreateCurrentScene(sceneInstanceId, clientScene.Zone, sceneName, currentScenesCom
        UnitComponent unitComponent = currentScene.AddComponent<UnitComponent>(); // <<<<<<<<<<<<<<<<<<<< 添加组件

        // 可以订阅这个事件中创建 Loading 界面
        EventSystem.Instance.Publish(clientScene, new EventType.SceneChangeStart());
        // 等待 CreateMyUnit 的消息
        Wait_CreateMyUnit waitCreateMyUnit = await clientScene.GetComponent<ObjectWait>().Wait<Wait_CreateMyUnit>();
        M2C_CreateMyUnit m2CCreateMyUnit = waitCreateMyUnit.Message;
        Unit unit = UnitFactory.Create(currentScene, m2CCreateMyUnit.Unit);
        unitComponent.Add(unit);

        clientScene.RemoveComponent<AIComponent>();

        EventSystem.Instance.Publish(currentScene, new EventType.SceneChangeFinish());
        // 通知等待场景切换的协程
        clientScene.GetComponent<ObjectWait>().Notify(new Wait_SceneChangeFinish());
    }
}
```

3.1.1 Unit: Unit 究竟是什么，干什么的？像是游戏的一个最小单位，有位置与旋转参数

```
[ChildOf(typeof(UnitComponent))]
[DebuggerDisplay("ViewName,nq")]
public class Unit: Entity, IAwake<int> {
    public int ConfigId { get; set; } // 配置表 id
    [BsonIgnore]
    public UnitConfig Config => UnitConfigCategory.Instance.Get(this.ConfigId);
    public UnitType Type => (UnitType)UnitConfigCategory.Instance.Get(this.ConfigId).Type;
    [BsonElement]
    private float3 position; // 坐标
    [BsonIgnore]
    public float3 Position {
        get => this.position;
        set {
            float3 oldPos = this.position;
            this.position = value;
            EventSystem.Instance.Publish(this.DomainScene(), new EventType.ChangePosition() { Unit = this, OldPos = oldPos
        }
    }
    [BsonIgnore]
    public float3 Forward {
        get => math.mul(this.Rotation, math.forward());
        set => this.Rotation = quaternion.LookRotation(value, math.up());
    }
    [BsonElement]
    private quaternion rotation;
    [BsonIgnore]
    public quaternion Rotation {
        get => this.rotation;
        set {
            this.rotation = value;
            EventSystem.Instance.Publish(this.DomainScene(), new EventType.ChangeRotation() { Unit = this });
        }
    }
    protected override string ViewName {
        get {
            return $"{this.GetType().Name} ({this.Id})";
        }
    }
}
```

3.1.2 UnitComponent: 组件

```
[ComponentOf(typeof(Scene))]
public class UnitComponent: Entity, IAwake, IDestroy {
}
```

3.1.3 UnitComponentSystem: 生成系. 感觉这个系统不太懂

```
[ObjectSystem]
public class UnitComponentAwakeSystem : AwakeSystem<UnitComponent> {
    protected override void Awake(UnitComponent self) {
    }
}
[ObjectSystem]
public class UnitComponentDestroySystem : DestroySystem<UnitComponent> {
    protected override void Destroy(UnitComponent self) {
    }
}
public static class UnitComponentSystem {
    public static void Add(this UnitComponent self, Unit unit) {
    }
    public static Unit Get(this UnitComponent self, long id) {
        Unit unit = self.GetChild<Unit>(id);
        return unit;
    }
    public static void Remove(this UnitComponent self, long id) {
        Unit unit = self.GetChild<Unit>(id);
        unit?.Dispose();
    }
}
```

3.1.4 UnitHelper: 帮助在不同使用情境下，拿到 unit

```
public static class UnitHelper {
    public static Unit GetMyUnitFromClientScene(Scene clientScene) {
        PlayerComponent playerComponent = clientScene.GetComponent<PlayerComponent>();
        Scene currentScene = clientScene.GetComponent<CurrentScenesComponent>().Scene;
        return currentScene.GetComponent<UnitComponent>().Get(playerComponent.MyId);
    }
    public static Unit GetMyUnitFromCurrentScene(Scene currentScene) {
        PlayerComponent playerComponent = currentScene.Parent.GetParent<Scene>().GetComponent<PlayerComponent>();
        return currentScene.GetComponent<UnitComponent>().Get(playerComponent.MyId);
    }
}
```

3.2 SceneFactory: ClientScene: 添加三组件：【CurrentScenesComponent】【PlayerComponent】【ObjectWait】。

- SceneChangeHelper 类会调用工厂加工。

```
public static class SceneFactory {
    public static async ETTask<Scene> CreateClientScene(int zone, string name) {
        await ETTask.CompletedTask;

        Scene clientScene = EntitySceneFactory.CreateScene(zone, SceneType.Client, name, ClientSceneManagerComponent.
        clientScene.AddComponent<CurrentScenesComponent>();// 它添加了这些组件，也看下
        clientScene.AddComponent<ObjectWait>();
        clientScene.AddComponent<PlayerComponent>();

        EventSystem.Instance.Publish(clientScene, new EventType.AfterCreateClientScene()); // 好奇葩的事件，去看下
        return clientScene;
    }
    public static Scene CreateCurrentScene(long id, int zone, string name, CurrentScenesComponent currentScenesCompon
        Scene currentScene = EntitySceneFactory.CreateScene(id, IdGenerater.Instance.GenerateInstanceId(), zone, Scen
        currentScenesComponent.Scene = currentScene;

        EventSystem.Instance.Publish(currentScene, new EventType.AfterCreateCurrentScene());
        return currentScene;
    }
}
```

3.2.1 UnitFactory: 为什么我抓出两个不一样的定义，还没弄明白

```
public static class UnitFactory {
    public static Unit Create(Scene scene, long id, UnitType unitType) {
        UnitComponent unitComponent = scene.GetComponent<UnitComponent>();
        switch (unitType) {
            case UnitType.Player: {
                Unit unit = unitComponent.AddChildWithId<Unit, int>(id, 1001);
                unit.AddComponent<MoveComponent>();
                unit.Position = new float3(-10, 0, -10);

                NumericComponent numericComponent = unit.AddComponent<NumericComponent>();
                numericComponent.Set(NumericType.Speed, 6f); // 速度是 6 米每秒
                numericComponent.Set(NumericType.AOI, 15000); // 视野 15 米

                unitComponent.Add(unit);
                // 加入 aoi
                unit.AddComponent<AOIEntity, int, float3>(9 * 1000, unit.Position);
                return unit;
            }
            default:
                throw new Exception($"not such unit type: {unitType}");
        }
    }
}
public static class UnitFactory {
    public static Unit Create(Scene currentScene, UnitInfo unitInfo) {
        UnitComponent unitComponent = currentScene.GetComponent<UnitComponent>();
        Unit unit = unitComponent.AddChildWithId<Unit, int>(unitInfo.UnitId, unitInfo.ConfigId);
        unitComponent.Add(unit);

        unit.Position = unitInfo.Position;
        unit.Forward = unitInfo.Forward;

        NumericComponent numericComponent = unit.AddComponent<NumericComponent>();
        foreach (var kv in unitInfo.KV) {
            numericComponent.Set(kv.Key, kv.Value);
        }

        unit.AddComponent<MoveComponent>();
        if (unitInfo.MoveInfo != null) {
            if (unitInfo.MoveInfo.Points.Count > 0) {
                unitInfo.MoveInfo.Points[0] = unit.Position;
                unit.MoveToAsync(unitInfo.MoveInfo.Points).Coroutine();
            }
        }
        unit.AddComponent<ObjectWait>();
        unit.AddComponent<XunLuoPathComponent>();

        EventSystem.Instance.Publish(unit.DomainScene(), new EventType.AfterUnitCreate() {Unit = unit});
        return unit;
    }
}
```

4 标签系: 标签系统重构了，现分为几个类型

4.1 ComponentOfAttribute : Attribute

```
// 组件类父级实体类型约束
// 父级实体类型唯一的 标记指定父级实体类型【ComponentOf(typeof(parentType)】
// 不唯一则标记【ComponentOf]
[AttributeUsage(AttributeTargets.Class)]
public class ComponentOfAttribute : Attribute {
    public Type Type;
    public ComponentOfAttribute(Type type = null) {
        this.Type = type;
    }
}
```

4.2 ComponentView: MonoBehaviour

```
public class ComponentView: MonoBehaviour {
    public Entity Component {
        get;
        set;
    }
}
```

4.3 ComponentViewEditor: Editor

```
[CustomEditor(typeof (ComponentView))]
public class ComponentViewEditor: Editor {
    public override void OnInspectorGUI() {
        ComponentView componentView = (ComponentView) target;
        Entity component = componentView.Component;
        ComponentViewHelper.Draw(component);
    }
}
```

5 UI 上的事件驱动系统:

5.1 EventType

```
namespace EventType {
    public struct SceneChangeStart {
    }
    public struct SceneChangeFinish {
    }

    public struct AfterCreateClientScene {
    }
    public struct AfterCreateCurrentScene {
    }

    public struct AppStartInitFinish {
    }
    public struct LoginFinish {
    }
    // public struct EnterMapFinish {
    public struct EnterRoomFinish {
    }
    public struct AfterUnitCreate {
        public Unit Unit;
    }
}
```

5.2 由 AppStartInitFinish 事件所触发的 CreateLoginUI

```
[Event(SceneType.Client)] // ET 事件系统的工具, 标签系
public class AppStartInitFinish_CreateLoginUI: AEvent<EventType.AppStartInitFinish> {
```

5.3 由 LoginFinish 事件所触发的 CreateLobbyUI

```
[Event(SceneType.Client)]
public class LoginFinish_CreateLobbyUI: AEvent<EventType.LoginFinish> {
    protected override async ETTask Run(Scene scene, EventType.LoginFinish args) {
        await UIHelper.Create(scene, UIType.UILobby, UILayer.Mid);
    }
}
```

- 这些是原示范框架都已经完成了的，我只需要添加剩余的逻辑。

5.4 SceneChangeStart_AddComponent: 开始切换场景的时候，就自动添加【OperaComponent】组件。现在对场景这块儿还不够熟悉

```
// 这个比较喜欢：场景切换，切换开始，可以做点什么？切换结束，可以做点什么？全成事件触发机制。任何时候，活宝妹就是一定要嫁给亲爱的表哥
[Event(SceneType.Client)]
public class SceneChangeStart_AddComponent: AEvent<EventType.SceneChangeStart> {
    protected override async ETTask Run(Scene scene, EventType.SceneChangeStart args) {
        Scene currentScene = scene.CurrentScene();
        // 加载场景资源
        await ResourcesComponent.Instance.LoadBundleAsync($"{currentScene.Name}.unity3d");
        // 切换到 map 场景
        await SceneManager.LoadSceneAsync(currentScene.Name);

        currentScene.AddComponent<OperaComponent>();
    }
}
```

- 场景加载结束的时候，好像相对做的事情不多。

6 Helper 类的总结：【但凡点击回调方法，就变成 Helper 类!】为什么就变成了这么一个个的帮助类呢？

6.1 LoginHelper.cs

```
public static class LoginHelper {
public static async ETTask Login(Scene clientScene, string account, string password) {
    try {
        // 创建一个 ETModel 层的 Session
        clientScene.RemoveComponent<RouterAddressComponent>();
        // 获取路由跟 realmDispatcher 地址
        RouterAddressComponent routerAddressComponent = clientScene.GetComponent<RouterAddressComponent>();
        if (routerAddressComponent == null) {
            routerAddressComponent = clientScene.AddComponent<RouterAddressComponent, string, int>(ConstValue.RouterHttpHos
            await routerAddressComponent.Init();

            clientScene.AddComponent<NetClientComponent, AddressFamily>(routerAddressComponent.RouterManagerIPAddress.Addre
        }
        IPEndPoint realmAddress = routerAddressComponent.GetRealmAddress(account);

        R2C_Login r2CLogin;
        using (Session session = await RouterHelper.CreateRouterSession(clientScene, realmAddress)) {
            r2CLogin = (R2C_Login) await session.Call(new C2R_Login() { Account = account, Password = password });
        }
        // 创建一个 gate Session，并且保存到 SessionComponent 中：与网关服的会话框。主要负责用户下线后会话框的自动移除销毁
        Session gateSession = await RouterHelper.CreateRouterSession(clientScene, NetworkHelper.ToIPEndPoint(r2CLogin.Addre
        clientScene.AddComponent<SessionComponent>().Session = gateSession;

        G2C_LoginGate g2CLoginGate = (G2C_LoginGate)await gateSession.Call(
            new C2G_LoginGate() { Key = r2CLogin.Key, GateId = r2CLogin.GateId});
        Log.Debug(" 登陆 gate 成功!");
        await EventSystem.Instance.PublishAsync(clientScene, new EventType.LoginFinish());
    }
    catch (Exception e) {
        Log.Error(e);
    }
}
}
```

6.2 EnterRoomHelper.cs

- 这里需要注意的是：原项目里面还是保留了 C2G_EnterMap 消息的。分两块查看一下：
  - 可以先去查一下，斗地主里是如何【开始匹配】的
  - ET 7 框架里，服务器是如何处理消息的，变成了不同的 **场景类型：SceneType，由不同场景，也就是不同的专职服务器来处理各种逻辑功能块的消息**

    * 仍然是 **标签系的消息处理器**: 因为先前的不同服变成了现在的不同场景，分场景（先前的不同服）来定义消息处理器，以处理当前场景（特定功能逻辑服）下的消息，如匹配服的消息。
  - **如果每个按钮的回调：都单独一个类，不成了海量回调类了？**
  - 老版本：斗地主里，进入地图的参考 **【ET】里，就要去找，如何处理这些组件的？**

```
// public static class EnterMapHelper {
public static class EnterRoomHelper {

// 进拖拉拉机房: 异步过程, 需要与房间服交互的. 【房间服】:
// 【C2G_EnterRoom】: 消息也改下
public static async ETTask EnterRoomAsync(Scene clientScene) {
    try {
        G2C_EnterMap g2CEnterMap = await clientScene.GetComponent<SessionComponent>().Session.Call(new C2G_EnterMap()) as G
        clientScene.GetComponent<PlayerComponent>().MyId = g2CEnterMap.MyId;

        // 等待场景切换完成
        await clientScene.GetComponent<ObjectWait>().Wait<Wait_SceneChangeFinish>();

        // EventSystem.Instance.Publish(clientScene, new EventType.EnterMapFinish());
        EventSystem.Instance.Publish(clientScene, new EventType.EnterRoomFinish()); // 这个, 再去找下, 谁在订阅这个事件, 如何

        // // 老版本: 斗地主里, 进入地图的参考【ET7】里, 就要去找, 如何处理这些组件的?
        // Game.Scene.AddComponent<OperaComponent>();
        // Game.Scene.GetComponent<UIComponent>().Remove(UIType.UILobby);
    }
    catch (Exception e) {
        Log.Error(e);
    }
}
}
```

- 一个服务器端的消息处理器供自己参考：【分场景的消息处理器，仍使用标签系】

```
[MessageHandler(SceneType.Client)]
public class M2C_CreateMyUnitHandler : AMHandler<M2C_CreateMyUnit> {
    protected override async ETTask Run(Session session, M2C_CreateMyUnit message) {
        // 通知场景切换协程继续往下走
        session.DomainScene().GetComponent<ObjectWait>().Notify(new Wait_CreateMyUnit() {Message = message});
        await ETTask.CompletedTask;
    }
}
```

- 再来一个场景切换开始事件的：【任何时候，活宝妹就是一定要嫁给亲爱的表哥!!!】

```
// 这个比较喜欢: 场景切换, 先前不同功能定义的服, 切换开始, 可以做点什么? 切换结束, 可以做点什么? 全成事件触发机制。
[Event(SceneType.Client)]
public class SceneChangeStart_AddComponent: AEvent<EventType.SceneChangeStart> {

    protected override async ETTask Run(Scene scene, EventType.SceneChangeStart args) {
        Scene currentScene = scene.CurrentScene();

        // 加载场景资源
        await ResourcesComponent.Instance.LoadBundleAsync($"{currentScene.Name}.unity3d");
        // 切换到 map 场景
        await SceneManager.LoadSceneAsync(currentScene.Name);

        currentScene.AddComponent<OperaComponent>();
    }
}
```

6.3 UIHelper.cs: 负责 UI 界面上的组件的，添加与删除，异步完成

```
public static class UIHelper {
    public static async ETTask<UI> Create(Scene scene, string uiType, UILayer uiLayer) {
        return await scene.GetComponent<UIComponent>().Create(uiType, uiLayer);
    }
    public static async ETTask Remove(Scene scene, string uiType) {
        scene.GetComponent<UIComponent>().Remove(uiType);
        await ETTask.CompletedTask;
    }
}
```

6.4 SceneChangeHelper: 场景切换协程

```
public static class SceneChangeHelper {
    // 场景切换协程
    public static async ETTask SceneChangeTo(Scene clientScene, string sceneName, long sceneInstanceId) {
        clientScene.RemoveComponent<AIComponent>();

        CurrentScenesComponent currentScenesComponent = clientScene.GetComponent<CurrentScenesComponent>();
        currentScenesComponent.Scene?.Dispose(); // 删除之前的 CurrentScene, 创建新的
        Scene currentScene = SceneFactory.CreateCurrentScene(sceneInstanceId, clientScene.Zone, sceneName, currentScenesCom
        UnitComponent unitComponent = currentScene.AddComponent<UnitComponent>();

        // 可以订阅这个事件中创建 Loading 界面
        EventSystem.Instance.Publish(clientScene, new EventType.SceneChangeStart());
        // 等待 CreateMyUnit 的消息
        Wait_CreateMyUnit waitCreateMyUnit = await clientScene.GetComponent<ObjectWait>().Wait<Wait_CreateMyUnit>();
        M2C_CreateMyUnit m2CCreateMyUnit = waitCreateMyUnit.Message;
        Unit unit = UnitFactory.Create(currentScene, m2CCreateMyUnit.Unit);
        unitComponent.Add(unit);

        clientScene.RemoveComponent<AIComponent>();

        EventSystem.Instance.Publish(currentScene, new EventType.SceneChangeFinish());
        // 通知等待场景切换的协程
        clientScene.GetComponent<ObjectWait>().Notify(new Wait_SceneChangeFinish());
    }
}
```

7 UI 控件的生产事件机制流程：以前的专用工厂再包装为 UI 上的事件机制

- 一般是由某个事件的发布，因为订阅（使用订阅标签系），所以会被触发创建视图

7.1 LoginHelper 发布 EventType.LoginFinish() 事件

```
public static class LoginHelper {
public static async ETTask Login(Scene clientScene, string account, string password) {
    try {
        // 创建一个 ETModel 层的 Session
        clientScene.RemoveComponent<RouterAddressComponent>();
        // 获取路由跟 realmDispatcher 地址
        RouterAddressComponent routerAddressComponent = clientScene.GetComponent<RouterAddressComponent>();
        if (routerAddressComponent == null) {
            routerAddressComponent = clientScene.AddComponent<RouterAddressComponent, string, int>(ConstValue.RouterHttpHos
            await routerAddressComponent.Init();
            clientScene.AddComponent<NetClientComponent, AddressFamily>(routerAddressComponent.RouterManagerIPAddress.Addre
        }
        IPEndPoint realmAddress = routerAddressComponent.GetRealmAddress(account);

        R2C_Login r2CLogin;
        using (Session session = await RouterHelper.CreateRouterSession(clientScene, realmAddress)) {
            r2CLogin = (R2C_Login) await session.Call(new C2R_Login() { Account = account, Password = password });
        }
        // 创建一个 gate Session, 并且保存到 SessionComponent 中: 与网关服的会话框。主要负责用户下线后会话框的自动移除销毁
        Session gateSession = await RouterHelper.CreateRouterSession(clientScene, NetworkHelper.ToIPEndPoint(r2CLogin.Addre
        clientScene.AddComponent<SessionComponent>().Session = gateSession;

        G2C_LoginGate g2CLoginGate = (G2C_LoginGate)await gateSession.Call(
            new C2G_LoginGate() { Key = r2CLogin.Key, GateId = r2CLogin.GateId});
        Log.Debug("登陆 gate 成功!");
        await EventSystem.Instance.PublishAsync(clientScene, new EventType.LoginFinish()); // <<<<<<<<<<<<<<<<<<<< 事件的发
    }
    catch (Exception e) {
        Log.Error(e);
    }
}
```

7.2 LoginFinish_RemoveLoginUI: 一般对应两个事件，旧视图的去除，与新视图的添加

```
[Event(SceneType.Client)]
public class LoginFinish_RemoveLoginUI: AEvent<EventType.LoginFinish> {
    protected override async ETTask Run(Scene scene, EventType.LoginFinish args) {
        await UIHelper.Remove(scene, UIType.UILogin);
    }
}
```

7.3 LoginFinish_CreateLobbyUI: 创建新视图

```
[Event(SceneType.Client)]
public class LoginFinish_CreateLobbyUI: AEvent<EventType.LoginFinish> {

    protected override async ETTask Run(Scene scene, EventType.LoginFinish args) {
        await UIHelper.Create(scene, UIType.UILobby, UILayer.Mid);
    }
}
```

7.4 UIHelper: 帮助类，来添加或是移除 UI 上的可装可折的组件

```
public static class UIHelper {
    public static async ETTask<UI> Create(Scene scene, string uiType, UILayer uiLayer) {
        return await scene.GetComponent<UIComponent>().Create(uiType, uiLayer); // <<<<<<<<<<<<<<<<<<<< 进一步调用
    }
    public static async ETTask Remove(Scene scene, string uiType) {
        scene.GetComponent<UIComponent>().Remove(uiType);
        await ETTask.CompletedTask;
    }
}
```

7.5 UIComponentSystem: 管理 Scene 上的 UI: 这个是组件生成管理系统，负责添加与删除。【UIEventComponent】是 UI 上的 UI 事件组件系统

```
// 管理 Scene 上的 UI: 这个是组件生成管理系统，负责添加与删除。【UIEventComponent】是 UI 上的 UI 事件组件系统
[FriendOf(typeof(UIComponent))]
public static class UIComponentSystem {
    public static async ETTask<UI> Create(this UIComponent self, string uiType, UILayer uiLayer) {
        UI ui = await UIEventComponent.Instance.OnCreate(self, uiType, uiLayer);  // <<<<<<<<<<<<<<<<<<<<
        self.UIs.Add(uiType, ui);
        return ui;
    }
    public static void Remove(this UIComponent self, string uiType) {
        if (!self.UIs.TryGetValue(uiType, out UI ui)) {
            return;
        }
        UIEventComponent.Instance.OnRemove(self, uiType);

        self.UIs.Remove(uiType);
        ui.Dispose();
    }
    public static UI Get(this UIComponent self, string name) {
        UI ui = null;
        self.UIs.TryGetValue(name, out ui);
        return ui;
    }
}
```

7.6 UIEventComponentSystem: 管理所有 UI GameObject 以及 UI 事件: 应该主要是 UI 控件相关的事件。【自顶向下】的组件系统

```
// 管理所有 UI GameObject 以及 UI 事件: 应该主要是 UI 控件相关的事件。【自顶向下】的组件系统
[FriendOf(typeof(UIEventComponent))]
public static class UIEventComponentSystem {
    [ObjectSystem]
    public class UIEventComponentAwakeSystem : AwakeSystem<UIEventComponent> {
        protected override void Awake(UIEventComponent self) {
            UIEventComponent.Instance = self;
            GameObject uiRoot = GameObject.Find("/Global/UI"); // Unity 视图面板上的全局父控件
            ReferenceCollector referenceCollector = uiRoot.GetComponent<ReferenceCollector>();
            // 面板上的: 四大层级
            self.UILayers.Add((int)UILayer.Hidden, referenceCollector.Get<GameObject>(UILayer.Hidden.ToString()).transform)
            self.UILayers.Add((int)UILayer.Low, referenceCollector.Get<GameObject>(UILayer.Low.ToString()).transform);
            self.UILayers.Add((int)UILayer.Mid, referenceCollector.Get<GameObject>(UILayer.Mid.ToString()).transform);
            self.UILayers.Add((int)UILayer.High, referenceCollector.Get<GameObject>(UILayer.High.ToString()).transform);
            var uiEvents = EventSystem.Instance.GetTypes(typeof (UIEventAttribute));
            foreach (Type type in uiEvents) {
                object[] attrs = type.GetCustomAttributes(typeof(UIEventAttribute), false);
                if (attrs.Length == 0) {
                    continue;
                }
                UIEventAttribute uiEventAttribute = attrs[0] as UIEventAttribute;
                // 字典管理: 它的字典, 负责为每种类型, 创建一个工厂实例, 来生产其所负责的 UI 组件面板等。字典管理, 工厂是可以随需要
                AUIEvent aUIEvent = Activator.CreateInstance(type) as AUIEvent;
                self.UIEvents.Add(uiEventAttribute.UIType, aUIEvent);
            }
        }
    }
    public static async ETTask<UI> OnCreate(this UIEventComponent self, UIComponent uiComponent, string uiType, UILayer uiL
        try {
            UI ui = await self.UIEvents[uiType].OnCreate(uiComponent, uiLayer); // 调用: 工厂的生产方法  // <<<<<<<<<<<<<<<<<<
            return ui;
        }
        catch (Exception e) {
            throw new Exception($"on create ui error: {uiType}", e);
        }
    }
    public static Transform GetLayer(this UIEventComponent self, int layer) {
        return self.UILayers[layer];
    }
    public static void OnRemove(this UIEventComponent self, UIComponent uiComponent, string uiType) {
        try {
            self.UIEvents[uiType].OnRemove(uiComponent);
        }
        catch (Exception e) {
            throw new Exception($"on remove ui error: {uiType}", e);
        }

    }
}
```

7.7 AUIEvent: 跟下面的 UIEventComponent 关系是？

```
public abstract class AUIEvent {
    public abstract ETTask<UI> OnCreate(UIComponent uiComponent, UILayer uiLayer);
    public abstract void OnRemove(UIComponent uiComponent);
}
```

7.8 UIEventComponent: 管理所有 UI GameObject

```
// 管理所有 UI GameObject
[ComponentOf(typeof(Scene))]
public class UIEventComponent: Entity, IAwake {
    [StaticField]
    public static UIEventComponent Instance;
    public Dictionary<string, AUIEvent> UIEvents = new Dictionary<string, AUIEvent>();
    public Dictionary<int, Transform> UILayers = new Dictionary<int, Transform>();
}
```

7.9 UIEventComponentSystem: 生成系, 管理所有 UI GameObject 以及 UI 事件: 应该主要是 UI 控件相关的事件。【自顶向下】的组件系统

```
// 管理所有 UI GameObject 以及 UI 事件: 应该主要是 UI 控件相关的事件。【自顶向下】的组件系统
[FriendOf(typeof(UIEventComponent))]
public static class UIEventComponentSystem {
    [ObjectSystem]
    public class UIEventComponentAwakeSystem : AwakeSystem<UIEventComponent> {
        protected override void Awake(UIEventComponent self) {
            UIEventComponent.Instance = self;
            GameObject uiRoot = GameObject.Find("/Global/UI"); // Unity 视图面板上的全局父控件
            ReferenceCollector referenceCollector = uiRoot.GetComponent<ReferenceCollector>();
            // 面板上的: 四大层级
            self.UILayers.Add((int)UILayer.Hidden, referenceCollector.Get<GameObject>(UILayer.Hidden.ToString()).transform)
            self.UILayers.Add((int)UILayer.Low, referenceCollector.Get<GameObject>(UILayer.Low.ToString()).transform);
            self.UILayers.Add((int)UILayer.Mid, referenceCollector.Get<GameObject>(UILayer.Mid.ToString()).transform);
            self.UILayers.Add((int)UILayer.High, referenceCollector.Get<GameObject>(UILayer.High.ToString()).transform);
            var uiEvents = EventSystem.Instance.GetTypes(typeof (UIEventAttribute));
            foreach (Type type in uiEvents) {
                object[] attrs = type.GetCustomAttributes(typeof(UIEventAttribute), false);
                if (attrs.Length == 0) {
                    continue;
                }
                UIEventAttribute uiEventAttribute = attrs[0] as UIEventAttribute;
                // 字典管理: 它的字典, 负责为每种类型, 创建一个工厂实例, 来生产其所负责的 UI 组件面板等。字典管理, 工厂是可以随需要
                AUIEvent aUIEvent = Activator.CreateInstance(type) as AUIEvent;
                self.UIEvents.Add(uiEventAttribute.UIType, aUIEvent);
            }
        }
    }
    public static async ETTask<UI> OnCreate(this UIEventComponent self, UIComponent uiComponent, string uiType, UILayer uiL
        try {
            UI ui = await self.UIEvents[uiType].OnCreate(uiComponent, uiLayer); // 调用: 工厂的生产方法
            return ui;
        }
        catch (Exception e) {
            throw new Exception($"on create ui error: {uiType}", e);
        }
    }
    public static Transform GetLayer(this UIEventComponent self, int layer) {
        return self.UILayers[layer];
    }
    public static void OnRemove(this UIEventComponent self, UIComponent uiComponent, string uiType) {
        try {
            self.UIEvents[uiType].OnRemove(uiComponent);
        }
        catch (Exception e) {
            throw new Exception($"on remove ui error: {uiType}", e);
        }
    }
}
```

7.10 UILoginEvent: 一个实体类的例子，具体的工厂生产逻辑

```
[UIEvent(UIType.UILogin)]
public class UILoginEvent: AUIEvent {
    public override async ETTask<UI> OnCreate(UIComponent uiComponent, UILayer uiLayer) {
        await uiComponent.DomainScene().GetComponent<ResourcesLoaderComponent>().LoadAsync(UIType.UILogin.StringToAB());
        GameObject bundleGameObject = (GameObject) ResourcesComponent.Instance.GetAsset(UIType.UILogin.StringToAB(), UIType
        GameObject gameObject = UnityEngine.Object.Instantiate(bundleGameObject, UIEventComponent.Instance.GetLayer((int)ui
        UI ui = uiComponent.AddChild<UI, string, GameObject>(UIType.UILogin, gameObject);
        ui.AddComponent<UILoginComponent>();
        return ui;
    }
    public override void OnRemove(UIComponent uiComponent) {
        ResourcesComponent.Instance.UnloadBundle(UIType.UILogin.StringToAB());
    }
}
```

7.11 UILobbyEvent: 再加一个实体类的例子

```
// UI 系统的事件机制: 接收到 LoginFinish 之后触发的大厅创建
[UIEvent(UIType.UILobby)]
public class UILobbyEvent: AUIEvent {
    public override async ETTask<UI> OnCreate(UIComponent uiComponent, UILayer uiLayer) {
        await ETTask.CompletedTask;
        await uiComponent.DomainScene().GetComponent<ResourcesLoaderComponent>().LoadAsync(UIType.UILobby.StringToAB());
        GameObject bundleGameObject = (GameObject) ResourcesComponent.Instance.GetAsset(UIType.UILobby.StringToAB(), UIType
        GameObject gameObject = UnityEngine.Object.Instantiate(bundleGameObject, UIEventComponent.Instance.GetLayer((int)ui
        UI ui = uiComponent.AddChild<UI, string, GameObject>(UIType.UILobby, gameObject);
        ui.AddComponent<UILobbyComponent>();
        return ui;
    }
    public override void OnRemove(UIComponent uiComponent) {
        ResourcesComponent.Instance.UnloadBundle(UIType.UILobby.StringToAB());
    }
}
```

8 Session 相关：进行间通信

8.1 SessionComponent

```
[ComponentOf(typeof(Scene))]
public class SessionComponent: Entity, IAwake, IDestroy {
    public Session Session { get; set; }
}
```

8.2 SessionComponentDestroySystem: 【销毁系】: 只负责用户掉线，或是下线后的自动移除会话框

```
// 【销毁系】: 只负责用户掉线，或是下线后的自动移除会话框
public class SessionComponentDestroySystem: DestroySystem<SessionComponent> {
    protected override void Destroy(SessionComponent self) {
        self.Session?.Dispose();
    }
}
```

8.3 OperaComponentSystem: 一个拿会话框必消息的使用场景

```
[FriendOf(typeof(OperaComponent))]
public static class OperaComponentSystem { // 生命周期感知，生成系统
    [ObjectSystem]
    public class OperaComponentAwakeSystem : AwakeSystem<OperaComponent> {
        protected override void Awake(OperaComponent self) {
            self.mapMask = LayerMask.GetMask("Map");
        }
    }
    [ObjectSystem]
    public class OperaComponentUpdateSystem : UpdateSystem<OperaComponent> {
        protected override void Update(OperaComponent self) {
            if (Input.GetMouseButtonDown(1)) {
                Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
                RaycastHit hit;
                if (Physics.Raycast(ray, out hit, 1000, self.mapMask)) {
                    C2M_PathfindingResult c2MPathfindingResult = new C2M_PathfindingResult();
                    c2MPathfindingResult.Position = hit.point;
                    self.ClientScene().GetComponent<SessionComponent>().Session.Send(c2MPathfindingResult); // <<<<<<<<<<<<
                }
            }
            if (Input.GetKeyDown(KeyCode.R)) {
                CodeLoader.Instance.LoadHotfix();
                EventSystem.Instance.Load();
                Log.Debug("hot reload success!");
            }
            if (Input.GetKeyDown(KeyCode.T)) {
                C2M_TransferMap c2MTransferMap = new C2M_TransferMap();
                self.ClientScene().GetComponent<SessionComponent>().Session.Call(c2MTransferMap).Coroutine(); // <<<<<<<<<<
            }
        }
    }
}
```

9 Player: 玩家相关，添加的地方，以及使用【这里还是有点儿糊涂】

9.1 Player: 玩家

```
[ChildOf(typeof(PlayerComponent))]
public sealed class Player : Entity, IAwake<string> {
    public string Account { get; set; }
    public long UnitId { get; set; }
}
```

9.2 PlayerComponent:

```
[ComponentOf(typeof(Scene))]
public class PlayerComponent: Entity, IAwake {
    public long MyId { get; set; }
}
```

9.3 服务器端的 PlayerComponent

```
namespace ET.Server {
[ComponentOf(typeof(Scene))]
public class PlayerComponent : Entity, IAwake, IDestroy {
    public readonly Dictionary<long, Player> idPlayers = new Dictionary<long, Player>();
}
```

9.4 服务器端 PlayerComponentSystem

```
namespace ET.Server {
    [FriendOf(typeof(PlayerComponent))]
    public static class PlayerComponentSystem {
        public static void Add(this PlayerComponent self, Player player) {
            self.idPlayers.Add(player.Id, player);
        }
        public static Player Get(this PlayerComponent self, long id) {
            self.idPlayers.TryGetValue(id, out Player gamer);
            return gamer;
        }
        public static void Remove(this PlayerComponent self, long id) {
            self.idPlayers.Remove(id);
        }
        public static Player[] GetAll(this PlayerComponent self) {
            return self.idPlayers.Values.ToArray();
        }
    }
}
```

9.5 服务器端 SceneFactory-CreateServerScene 时【网关服】会添加【PlayerComponent】玩家组件

```
public static class SceneFactory {
    public static async ETTask<Scene> CreateServerScene(Entity parent, long id, long instanceId, int zone, string name, Sce
        await ETTask.CompletedTask;
        Scene scene = EntitySceneFactory.CreateScene(id, instanceId, zone, sceneType, name, parent);
        scene.AddComponent<MailBoxComponent, MailboxType>(MailboxType.UnOrderMessageDispatcher);
        switch (scene.SceneType) {
        case SceneType.Router:
            scene.AddComponent<RouterComponent, IPEndPoint, string>(startSceneConfig.OuterIPPort, startSceneConfig.StartPro
            break;
        case SceneType.RouterManager: // 正式发布请用 CDN 代替 RouterManager
            // 云服务器在防火墙那里做端口映射
            scene.AddComponent<HttpComponent, string>($"http:// *:{startSceneConfig.OuterPort}/");
            break;
        case SceneType.Realm:
            scene.AddComponent<NetServerComponent, IPEndPoint>(startSceneConfig.InnerIPOutPort);
            break;
        case SceneType.Gate: // <<<<<<<<<<<<<<<<<<<<
            scene.AddComponent<NetServerComponent, IPEndPoint>(startSceneConfig.InnerIPOutPort);
            scene.AddComponent<PlayerComponent>(); // <<<<<<<<<<<<<<<<<<<<
            scene.AddComponent<GateSessionKeyComponent>();
            break;
        case SceneType.Map:
            scene.AddComponent<UnitComponent>();
            scene.AddComponent<AOIManagerComponent>();
            break;
        case SceneType.Location:
            scene.AddComponent<LocationComponent>();
            break;
        case SceneType.Robot:
            scene.AddComponent<RobotManagerComponent>();
            break;
        case SceneType.BenchmarkServer:
            scene.AddComponent<BenchmarkServerComponent>();
            scene.AddComponent<NetServerComponent, IPEndPoint>(startSceneConfig.OuterIPPort);
            break;
        case SceneType.BenchmarkClient:
            scene.AddComponent<BenchmarkClientComponent>();
            break;
        }
        return scene;
    }
```

9.6 SessionPlayerComponentSystem

```
public static class SessionPlayerComponentSystem {
    public class SessionPlayerComponentDestroySystem: DestroySystem<SessionPlayerComponent> {
        protected override void Destroy(SessionPlayerComponent self) {
            // 发送断线消息
            ActorLocationSenderComponent.Instance?.Send(self.PlayerId, new G2M_SessionDisconnect());
            self.DomainScene().GetComponent<PlayerComponent>()?.Remove(self.PlayerId);
        }
    }
    public static Player GetMyPlayer(this SessionPlayerComponent self) {
        return self.DomainScene().GetComponent<PlayerComponent>().Get(self.PlayerId);
    }
}
```

9.7 SessionPlayerComponent: 会话框里，会保留客户端玩家 PlayerId

```
[ComponentOf(typeof(Session))]
public class SessionPlayerComponent : Entity, IAwake, IDestroy {
    public long PlayerId { get; set; }
}
```

10 ResourcesComponent 资源包管理器相关：有时候，拖拉机游戏里会需要拿它来加载图片

10.1 ResourcesComponent: 同文件有其生成系

```
[ComponentOf]
public class ResourcesComponent: Entity, IAwake, IDestroy {
    public static ResourcesComponent Instance { get; set; }
    public AssetBundleManifest AssetBundleManifestObject { get; set; }
    public Dictionary<int, string> IntToStringDict = new Dictionary<int, string>();
    public Dictionary<string, string> StringToABDict = new Dictionary<string, string>();
    public Dictionary<string, string> BundleNameToLowerDict = new Dictionary<string, string>() { { "StreamingAssets", "Stre
    public readonly Dictionary<string, Dictionary<string, UnityEngine.Object>> resourceCache =
        new Dictionary<string, Dictionary<string, UnityEngine.Object>>();
    public readonly Dictionary<string, ABInfo> bundles = new Dictionary<string, ABInfo>();
    // 缓存包依赖，不用每次计算
    public readonly Dictionary<string, string[]> DependenciesCache = new Dictionary<string, string[]>();
}
```

10.2 客户端 ConfigLoader 的 Invoke 标签下：在根控件 Root 下添加资源管理器组件

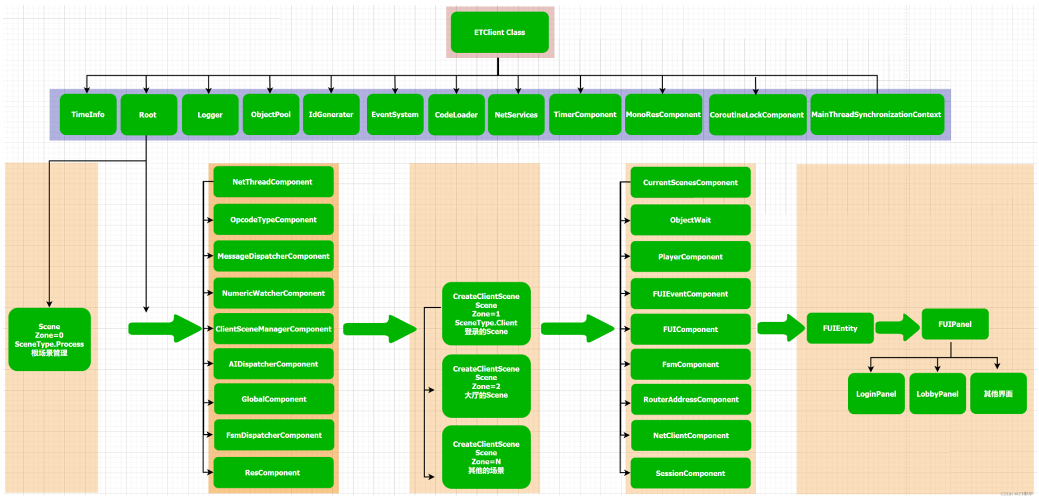
```
[Invoke]
public class GetAllConfigBytes: AInvokeHandler<ConfigComponent.GetAllConfigBytes, Dictionary<Type, byte[]>> {
    public override Dictionary<Type, byte[]> Handle(ConfigComponent.GetAllConfigBytes args) {
        Dictionary<Type, byte[]> output = new Dictionary<Type, byte[]>();
        HashSet<Type> configTypes = EventSystem.Instance.GetTypes(typeof (ConfigAttribute));

        if (Define.IsEditor) {
            string ct = "cs";
            GlobalConfig globalConfig = Resources.Load<GlobalConfig>("GlobalConfig");
            CodeMode codeMode = globalConfig.CodeMode;
            switch (codeMode) {
            case CodeMode.Client:
                ct = "c";
                break;
            case CodeMode.Server:
                ct = "s";
                break;
            case CodeMode.ClientServer:
                ct = "cs";
                break;
            default:
                throw new ArgumentOutOfRangeException();
            }
            List<string> startConfigs = new List<string>() {
                "StartMachineConfigCategory",
                "StartProcessConfigCategory",
                "StartSceneConfigCategory",
                "StartZoneConfigCategory",
            };
            foreach (Type configType in configTypes) {
                string configFilePath;
                if (startConfigs.Contains(configType.Name)) {
                    configFilePath = $"../Config/Excel/{ct}/{Options.Instance.StartConfig}/{configType.Name}.bytes";
                }
                else {
                    configFilePath = $"../Config/Excel/{ct}/{configType.Name}.bytes";
                }
                output[configType] = File.ReadAllBytes(configFilePath);
            }
        } else {
            using (Root.Instance.Scene.AddComponent<ResourcesComponent>()) { // <<<<<<<<<<<<<<<<<<<<
                const string configBundleName = "config.unity3d";
                ResourcesComponent.Instance.LoadBundle(configBundleName);

                foreach (Type configType in configTypes) {
                    TextAsset v = ResourcesComponent.Instance.GetAsset(configBundleName, configType.Name) as TextAsset;
                    output[configType] = v.bytes;
                }
            }
        }
        return output;
    }
}
```

11 整个框架：ET 7.2 + YooAssets + luban + FairGUI

- 整个框架的场景节点如下



12 写在最后：反而是自己每天查看一再更新的

- 因为感觉还是不曾系统性地读 ET7 的源码，或者说有效阅读，因为没有带着实际问题的看源码，感觉都不叫看读源码呀。这里会记自己的感觉需要赶快查看的地方。
- 【ET 框架的整体架构】：感觉把握不够。常常命名空间分不清。要把这个大的框架，比较高层面的架构再好好看下
- 然后就是对自顶向下的不同层级场景，所需要的主要的不同组件，分不清，仍需要再熟悉一下源码
- 【问题】：某些消息，还分不清是内网还是外网消息，暂时先放一下，到时再改
- 【问题】：上次那个 ET-EUI 框架的时候，曾经出现过 opcode 不对应，也就是说，我现在生成的进程间消息，有可能还是会存在服务器码与客户端码不对应，这个完备的框架，这次应该不至于吧？
- 【ClientComponent】：新框架里重构丢了，去找怎么替代？那么现在去追一下，客户端的起始与场景加载或是切换大致过程。它变成了什么客户端场景管理？
- 【UIType】部分类：这个类出现在了三四个不同的程序域，现在重构了，好像添加得不对。要再修改

13 现在的修改内容，记忆

- UILobbyComponent 里三个按钮的回调：这里面还有好几个错误。把这个弄完了，出错在更晚的地方的话，这个界面就可以加载完整了。。

```
// 获取玩家数据：按说应该是注册登录服的逻辑，或者是数据库服存放着用户信息，都是通过 Gate 中转
        long userId = ClientComponent.Instance.LocalPlayer.UserID; // 【ClientComponent】：组件被重构掉了，去找相应的替换
        C2G_GetUserInfo_Req c2G_GetUserInfo_Req = new C2G_GetUserInfo_Req() { UserID = userId }; // 去从网关服拿玩家信息
        G2C_GetUserInfo_Ack g2C_GetUserInfo_Ack = await SessionComponent.Instance.Session.Call(c2G_GetUserInfo_Req) as G2C_
        // 显示用户信息
        rc.Get<GameObject>("NickName").GetComponent<Text>().text = g2C_GetUserInfo_Ack.NickName;
        rc.Get<GameObject>("Money").GetComponent<Text>().text = g2C_GetUserInfo_Ack.Money.ToString();
    }
}
// 【回调:】自定义三个按钮的回调。这些个过程流程，就主要参考，同框架的斗地主游戏
public static async ETTask matchRoom(this UILobbyComponent self) { // 通过网关服中转，请求匹配服为给匹配一个房间四人桌
    try {
        // 发送开始匹配消息
        C2G_StartMatch_Req c2G_StartMatch_Req = new C2G_StartMatch_Req();
        G2C_StartMatch_Ack g2C_StartMatch_Ack = await SessionComponent.Instance.Session.Call(c2G_StartMatch_Req) as G2C_Sta
        // // 暂时跳过这步
        // if (g2C_StartMatch_Ack.Error == ErrorCode.ERR_UserMoneyLessError) {
        //     Log.Error(" 余额不足"); // 就是说，当且仅当余额不足的时候才会出这个错误？
        //     return;
        // }
        // 匹配成功了：UI 界面切换，切换到房间界面【UI 事件系统】：这里不再是手动添加与移除，去发布事件
        UI room = Game.Scene.GetComponent<UIComponent>().Create(UIType.LandlordsRoom); // 装载新的 UI 视图
        Game.Scene.GetComponent<UIComponent>().Remove(UIType.LandlordsLobby);          // 卸载旧的 UI 视图
        // 将房间设为匹配状态
        room.GetComponent<LandlordsRoomComponent>().Matching = true;
    }
    catch (Exception e) {
        Log.Error(e.ToStr());
    }
}
// 接下来，这两个选项，暂时不处理
public static async ETTask enterRoom(this UILobbyComponent self) { // 不知道，这个，与 EnterMap 有没有本质的区别，要检查一下
                            await EnterRoomHelper.EnterRoomAsync(self.ClientScene());
                                            await UIHelper.Remove(self.ClientScene(), UIType.UILobby);
                                            }
                                            public static async ETTask createRoom(this UILobbyComponent self) {

            }
```

- 【任何时候，活宝妹就是一定要嫁给亲爱的表哥!!!】
- 【活宝妹坐等亲爱的表哥，领娶活宝妹回家！爱表哥，爱生活!!!】

14 TODO 今天晚上把几个消息抓全了，免得一堆的报错

- IStart: 给重构没了。要重新熟悉一下新框架里的各种接口类，这些接口起什么作用，需不需要添加 IStart, 以及如何改写等
- 需要抓的消息包括：

C2G_ReturnLobby_Ntt Actor_GamerReady_Ntt Actor_Trusteeship_Ntt Actor_GamerPlayCard_Req