### ET 框架学习笔记(二) - - 网络交互相关

### deepwaterooo

May 18, 2023

### **Contents**

1	ET7 数据库相关【服务端】	1
	1.1 IDBCollection: 主要是方便写两个不同的数据库 (好像是 GeekServer 里两个数据库)。	
	反正方便扩展吧	
	1.2 DBComponent: 带生成系。可以查表,查询数据	
	1.3 DBManagerComponent: 有上面的 DBComponent 数组。数组长度固定吗?	
	1.4 DBManagerComponentSystem: 主是要查询某个区服的数据库,从数组里	
	1.5 DBProxyComponent: 【参考项目】里的。有生成系。	2
2	StartConfigComponent: 找【各种服】的起始初始化地址	2
	2.1 ConfigSingleton <t>: ProtoObject, ISingleton</t>	2
	2.2 SceneFactory 里可以给【匹配服】添加组件	
	2.3 RouterAddressComponent: 路由器组件	
	2.4 RouterAddressComponentSystem: 路由器的生成系	
	2.5 RouterHelper: 路由器帮助类,向路由器注册、申请?	
	2.6 StartProcessConfigCategory: ConfigSingleton <startprocessconfigcategory>, IMed</startprocessconfigcategory>	_
	【任何时候,活宝妹就是一定要嫁给亲爱的表哥!!!】	
	2.7 StartSceneConfig: ISupportInitialize 【各种服 - 配置,场景配置】	
	2.8 StartSceneConfigCategory: [Matchs!]ConfigSingleton <startsceneconfigcategory< th=""><th></th></startsceneconfigcategory<>	
	IMerge	6 7
	2.10 HttpGetRouterHandler: IHttpHandler: 获取各路由器的地址	
	2.11HttpHandler 标签系:标签自带场景类型	
	2.12LoginHelper: 登录服的获取地址的方式来获取匹配服的地址了。全框架只有这一个黄	U
	金案例	8
3	服务器的功能概述:各服务器的作用(这个不是 ET7 版本的,以前的)	9
4	Protobuf 相关,【Protobuf 里进程间传递的游戏数据相关信息:两个思路】	10
5	写在最后: 反而是自己每天查看一再更新的	10
6	现在的修改内容:	11
7	TODO 其它的:部分完成,或是待完成的大的功能版块,列举	11
8	梅拉机游戏:【重构 OOP/OOD 设计思路】	12

### 1 ET7 数据库相关【服务端】

- 这个数据库系统, 连个添加使用的范例也没有。。。就两个组件, 一个管理类。什么也没留下。。
- 这里不急着整理。现框架 **DB 放在服务端的 Model** 里。它的管理体系成为管理各个不同区服的数据库 DBComponent。
- 因为找不到任何参考使用的例子。我觉得需要搜索一下。在理解了参考项目数据库模块之后, 根据搜索,决定是使用原参考项目总服务器代理系,还是这种相对改装了的管理区服系统?
- 1.1 IDBCollection: 主要是方便写两个不同的数据库(好像是 GeekServer 里两个数据库)。反正方便扩展吧

```
public interface IDBCollection {}
```

1.2 DBComponent: 带生成系。可以查表, 查询数据

```
[ChildOf(typeof(DBManagerComponent))] // 用来缓存数据
public class DBComponent: Entity, IAwake<string, string, int>, IDestroy {
   public const int TaskCount = 32;
   public MongoClient mongoClient;
   public IMongoDatabase database;
}
```

1.3 DBManagerComponent: 有上面的 DBComponent 数组。数组长度固定吗?

```
public class DBManagerComponent: Entity, IAwake, IDestroy {
    [StaticField]
    public static DBManagerComponent Instance;
    public DBComponent[] DBComponents = new DBComponent[IdGenerater.MaxZone]; // 没事吃饱了撑得, 占一大堆空地
}
```

1.4 DBManagerComponentSystem: 主是要查询某个区服的数据库, 从数组里

```
[FriendOf(typeof(DBManagerComponent))]
public static class DBManagerComponentSystem {
    [ObjectSystem]
   public class DBManagerComponentAwakeSystem: AwakeSystem<DBManagerComponent> {
        protected override void Awake(DBManagerComponent self) {
           DBManagerComponent.Instance = self;
   }
    public class DBManagerComponentDestroySystem: DestroySystem<DBManagerComponent> {
       protected override void Destroy(DBManagerComponent self) {
           DBManagerComponent.Instance = null;
    public static DBComponent GetZoneDB(this DBManagerComponent self, int zone) {
       DBComponent dbComponent = self.DBComponents[zone];
        if (dbComponent != null)
            return dbComponent;
       StartZoneConfig startZoneConfig = StartZoneConfigCategory.Instance.Get(zone);
        if (startZoneConfig.DBConnection == "")
            throw new Exception($"zone: {zone} not found mongo connect string");
        dbComponent = self.AddChild<DBComponent, string, int>(startZoneConfig.DBConnection, startZoneConfig.DBName,
        self.DBComponents[zone] = dbComponent;
        return dbComponent;
   }
}
```

### 1.5 DBProxyComponent:【参考项目】里的。有生成系。

```
// 用来与数据库操作代理
public class DBProxyComponent: Component {
public IPEndPoint dbAddress;
}
```

### 2 StartConfigComponent: 找【各种服】的起始初始化地址

- 这些组群服务器的起始被全部重构了, 重构成配置单例了
- 这里是,昨天之前理这部分的时候,理得不够透彻。去重构一个服的初始化与配置,就需要理清、找到这个服所关联的所有的起始配置相关。昨天之前做得还不够好。
- 昨天晚上稍微理解得再深一点儿。应该也是,总结自己理一个相对大型框架的方法总结吧。
- 今天早上的脑袋还是相对比较清醒,就把这块儿对自己来说相对不够熟悉相对难点儿的,再 看一遍,希望有更多的收获!

#### 2.1 ConfigSingleton<T>: ProtoObject, ISingleton

```
public abstract class ConfigSingleton<T>: ProtoObject, ISingleton where T: ConfigSingleton<T>, new() {
        [StaticField]
        private static T instance;
       public static T Instance {
                return instance ??= ConfigComponent.Instance.LoadOneConfig(typeof (T)) as T;
       }
       void ISingleton.Register() {
            if (instance != null) {
                throw new Exception($"singleton register twice! {typeof (T).Name}");
           instance = (T)this;
        void ISingleton.Destroy() {
           T t = instance;
           instance = null:
           t.Dispose();
        bool ISingleton.IsDisposed() {
            throw new NotImplementedException();
        public override void AfterEndInit() { }
       public virtual void Dispose() { }
```

### 2.2 SceneFactory 里可以给【匹配服】添加组件

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<A0IManagerComponent>();
        break;
    case SceneType.Location:
        scene.AddComponent<LocationComponent>();
        break;
//...
} return scene;
}
```

#### 2.3 RouterAddressComponent: 路由器组件

```
[ComponentOf(typeof(Scene))]
public class RouterAddressComponent: Entity, IAwake<string, int> {
    public IPAddress RouterManagerIPAddress { get; set; }
    public string RouterManagerHost;
    public int RouterManagerPort;
    public HttpGetRouterResponse Info;
    public int RouterIndex;
}
```

### 2.4 RouterAddressComponentSystem: 路由器的生成系

```
[FriendOf(typeof(RouterAddressComponent))]
public static class RouterAddressComponentSystem {
    public class RouterAddressComponentAwakeSystem: AwakeSystemRouterAddressComponent, string, int> {
        protected override void Awake(RouterAddressComponent self, string address, int port) {
            self.RouterManagerHost = address;
            self.RouterManagerPort = port;
        }
   public static async ETTask Init(this RouterAddressComponent self) {
        self.RouterManagerIPAddress = NetworkHelper.GetHostAddress(self.RouterManagerHost);
        await self.GetAllRouter();
    private static async ETTask GetAllRouter(this RouterAddressComponent self) {
        string url = $"http:// {self.RouterManagerHost}:{self.RouterManagerPort}/get_router?v={RandomGenerator.RandUInt32()
       Log.Debug($"start get router info: {url}");
        string routerInfo = await HttpClientHelper.Get(url);
        Log.Debug($"recv router info: {routerInfo}");
       HttpGetRouterResponse httpGetRouterResponse = JsonHelper.FromJson<HttpGetRouterResponse>(routerInfo);
        self.Info = httpGetRouterResponse;
       Log.Debug($"start get router info finish: {JsonHelper.ToJson(httpGetRouterResponse)}");
        // 打乱顺序
       RandomGenerator.BreakRank(self.Info.Routers);
        self.WaitTenMinGetAllRouter().Coroutine();
    // 等 10 分钟再获取一次
    public static async ETTask WaitTenMinGetAllRouter(this RouterAddressComponent self) {
        await TimerComponent.Instance.WaitAsync(5 * 60 * 1000);
        if (self.IsDisposed)
           return;
        await self.GetAllRouter();
    public static IPEndPoint GetAddress(this RouterAddressComponent self) {
        if (self.Info.Routers.Count == 0)
            return null:
        string address = self.Info.Routers[self.RouterIndex++ % self.Info.Routers.Count];
        string[] ss = address.Split(':');
        IPAddress ipAddress = IPAddress.Parse(ss[0]);
        if (self.RouterManagerIPAddress.AddressFamily == AddressFamily.InterNetworkV6) {
            ipAddress = ipAddress.MapToIPv6();
        return new IPEndPoint(ipAddress, int.Parse(ss[1]));
```

```
public static IPEndPoint GetRealmAddress(this RouterAddressComponent self, string account) { // <<<<>>>>
        int v = account.Mode(self.Info.Realms.Count);
        string address = self.Info.Realms[v];
        string[] ss = address.Split(':');
        IPAddress ipAddress = IPAddress.Parse(ss[0]);
        // if (self.IPAddress.AddressFamily == AddressFamily.InterNetworkV6)
        // ipAddress = ipAddress.MapToIPv6();
        return new IPEndPoint(ipAddress, int.Parse(ss[1]));
    }
}
```

#### 2.5 RouterHelper: 路由器帮助类,向路由器注册、申请?

```
public static class RouterHelper {
      // 注册 router
      public static async ETTask<Session> CreateRouterSession(Scene clientScene, IPEndPoint address) {
              (uint recvLocalConn, IPEndPoint routerAddress) = await GetRouterAddress(clientScene, address, 0, 0);
             if (recvLocalConn == 0)
                    throw new Exception($"get router fail: {clientScene.Id} {address}");
             Log.Info($"get router: {recvLocalConn} {routerAddress}");
             Session\ router Session\ =\ client Scene. Get Component < Net Client Component > (). Create (router Address,\ address,\ recv Local Connucleon Component < (), Create (router Address),\ recv Local Connucleon C
             routerSession.AddComponent<PingComponent>();
             routerSession.AddComponent<RouterCheckComponent>();
             return routerSession;
      public static async ETTask<(uint, IPEndPoint)> GetRouterAddress(Scene clientScene, IPEndPoint address, uint localConn,
             Log.Info($"start get router address: {clientScene.Id} {address} {localConn} {remoteConn}");
             // return (RandomHelper.RandUInt32(), address);
             Router Address Component \ \ router Address Component = client Scene. \\ Get Component < Router Address Component > (); \\
             IPEndPoint routerInfo = routerAddressComponent.GetAddress();
             uint recvLocalConn = await Connect(routerInfo, address, localConn, remoteConn);
             Log.Info($"finish get router address: {clientScene.Id} {address} {localConn} {remoteConn} {recvLocalConn} {routerIn
             return (recvLocalConn, routerInfo);
      }
      // 向 router 申请
      private static async ETTask<uint> Connect(IPEndPoint routerAddress, IPEndPoint realAddress, uint localConn, uint remote
             uint connectId = RandomGenerator.RandUInt32();
             using Socket socket = new Socket(routerAddress.AddressFamily, SocketType.Dgram, ProtocolType.Udp);
             int count = 20;
             byte[] sendCache = new byte[512];
             byte[] recvCache = new byte[512];
             uint synFlag = localConn == 0? KcpProtocalType.RouterSYN : KcpProtocalType.RouterReconnectSYN;
             sendCache.WriteTo(0, synFlag);
             sendCache.WriteTo(1, localConn);
             sendCache.WriteTo(5, remoteConn);
             sendCache.WriteTo(9, connectId);
             byte[] addressBytes = realAddress.ToString().ToByteArray();
             Array.Copy(addressBytes, 0, sendCache, 13, addressBytes.Length);
             Log.Info($"router connect: {connectId} {localConn} {remoteConn} {routerAddress} {realAddress}");
             EndPoint recvIPEndPoint = new IPEndPoint(IPAddress.Any, 0);
             long lastSendTimer = 0;
             while (true) {
                    long timeNow = TimeHelper.ClientFrameTime();
                    if (timeNow - lastSendTimer > 300) {
                          if (--count < 0) {
                                 Log.Error($"router connect timeout fail! {localConn} {remoteConn} {routerAddress} {realAddress}");
                                 return 0:
                           lastSendTimer = timeNow;
                          // 发送
                           socket.SendTo(sendCache, 0, addressBytes.Length + 13, SocketFlags.None, routerAddress);
                    await TimerComponent.Instance.WaitFrameAsync();
                    // 接收
                    if (socket.Available > 0) {
                           int messageLength = socket.ReceiveFrom(recvCache, ref recvIPEndPoint);
                           if (messageLength != 9) {
                                 \textbf{Log.Error(\$"router connect error1: \{connectId\} \{messageLength\} \{localConn\} \{remoteConn\} \{routerAddress\} \}} \\
                                 continue:
                          byte flag = recvCache[0];
                           if (flag != KcpProtocalType.RouterReconnectACK && flag != KcpProtocalType.RouterACK) {
```

```
Log.Error($"router connect error2: {connectId} {synFlag} {flag} {localConn} {remoteConn} {routerAddress continue;} 
} uint recvRemoteConn = BitConverter.ToUInt32(recvCache, 1); 
uint recvLocalConn = BitConverter.ToUInt32(recvCache, 5); 
Log.Info($"router connect finish: {connectId} {recvRemoteConn} {recvLocalConn} {remoteConn} {router recvLocalConn; } 
} 
} 
}
```

# 2.6 StartProcessConfigCategory: ConfigSingleton<StartProcessConfigCateg IMerge: 【任何时候,活宝妹就是一定要嫁给亲爱的表哥!!!】

```
[ProtoContract]
[Config]
public partial class StartProcessConfigCategory : ConfigSingleton<StartProcessConfigCategory>, IMerge {
    [BsonIanore]
    private Dictionary<int, StartProcessConfig> dict = new Dictionary<int, StartProcessConfig>(); // 管理字典
    [BsonElement]
    [ProtoMember(1)]
    private List<StartProcessConfig> list = new List<StartProcessConfig>();
    public void Merge(object o) {
        StartProcessConfigCategory s = o as StartProcessConfigCategory;
        this.list.AddRange(s.list);
    [ProtoAfterDeserialization]
    public void ProtoEndInit() {
        foreach (StartProcessConfig config in list) {
            config.AfterEndInit();
            this.dict.Add(config.Id, config);
        this.list.Clear();
        this.AfterEndInit();
    public StartProcessConfig Get(int id) {
        this.dict.TryGetValue(id, out StartProcessConfig item);
        if (item == null) {
            throw new Exception($" 配置找不到, 配置表名: {nameof (StartProcessConfig)}, 配置 id: {id}");
        return item;
    public bool Contain(int id) {
        return this.dict.ContainsKey(id);
    public Dictionary<int, StartProcessConfig> GetAll() {
        return this.dict;
    }
    public StartProcessConfig GetOne() {
        if (this.dict == null || this.dict.Count <= 0) {</pre>
            return null;
        return this.dict.Values.GetEnumerator().Current;
    }
[ProtoContract]
public partial class StartProcessConfig: ProtoObject, IConfig {
    [ProtoMember(1)]
    public int Id { get; set; }
    [ProtoMember(2)]
    public int MachineId { get; set; }
    [ProtoMember(3)]
    public int InnerPort { get; set; }
}
```

### 2.7 StartSceneConfig: ISupportInitialize 【各种服 - 配置,场景配置】

```
public partial class StartSceneConfig: ISupportInitialize {
   public long InstanceId;
```

```
public SceneType Type; // 场景类型
    public StartProcessConfig StartProcessConfig {
           return StartProcessConfigCategory.Instance.Get(this.Process);
   public StartZoneConfig StartZoneConfig {
       get {
           return StartZoneConfigCategory.Instance.Get(this.Zone);
   }
    // 内网地址外网端口, 通过防火墙映射端口过来
   private IPEndPoint innerIPOutPort;
    public IPEndPoint InnerIPOutPort {
       get {
            if (innerIPOutPort == null) {
               this.innerIPOutPort = NetworkHelper.ToIPEndPoint($"{this.StartProcessConfig.InnerIP}:{this.OuterPort}");
           return this.innerIPOutPort:
    // 外网地址外网端口
    private IPEndPoint outerIPPort;
    public IPEndPoint OuterIPPort {
           if (this.outerIPPort == null) {
               this.outerIPPort = NetworkHelper.ToIPEndPoint($"{this.StartProcessConfig.OuterIP}:{this.OuterPort}");
           return this.outerIPPort:
        }
   }
    public override void AfterEndInit() {
        this.Type = EnumHelper.FromString<SceneType>(this.SceneType);
        InstanceIdStruct instanceIdStruct = new InstanceIdStruct(this.Process, (uint) this.Id);
        this.InstanceId = instanceIdStruct.ToLong();
   }
}
```

# 2.8 StartSceneConfigCategory: [Matchs!]ConfigSingleton<StartSceneConfi IMerge

- 读里面的登录服,会知道它是如何管理登录服的(就是后面的例子,当它要拿登录服的地址的时候),它们是区服,就是分各个小区管理。如果集群是这个样子,大概匹配服也就是一样分小区管理了。
- 那么这个配置管理里,因为我要用匹配服与地图服,也要对至少是匹配服进行管理。那么,我在申请匹配的时候,网关服才能拿到匹配服的地址。
- 只在【服务端】存在。但是在双端模式、与服务端模式下,每种端有两个文件来定义这个类。。 一个在【ProtoContract】里,可能可以进程间消息传递?一个在 ConfigPartial 文件夹里
- 上面的文件重复,还不是很懂。【重构】: 因为我现在还比较喜欢使用 Unity 下自带的双端模式,可是暂时只改【双端模式 ClientServer】下的文件,另一个专职服务端可能晚点儿再补上去。不用昨天晚上一样每个文件都改。

```
// 配置文件处理,或是服务器启动相关类,以前都没仔细读过
public partial class StartSceneConfigCategory {
    public MultiMap<int, StartSceneConfig> Gates = new MultiMap<int, StartSceneConfig>();
    public MultiMap<int, StartSceneConfig> ProcessScenes = new MultiMap<int, StartSceneConfig>();
    public Dictionary<long, Dictionary<string, StartSceneConfig>> ClientScenesByName = new Dictionary<long, Dictionary<string
    public StartSceneConfig LocationConfig;
    public List<StartSceneConfig> Realms = new List<StartSceneConfig>();
    public List<StartSceneConfig> Matchs = new List<StartSceneConfig>(); // <<<<<> 添加管理
    public List<StartSceneConfig> Routers = new List<StartSceneConfig>();
    public StartSceneConfig> Robots = new List<StartSceneConfig>();
    public StartSceneConfig> Robots = new List<StartSceneConfig>();
    public StartSceneConfig BenchmarkServer;
```

```
public List<StartSceneConfig> GetByProcess(int process) {
   return this.ProcessScenes[process];
public StartSceneConfig GetBySceneName(int zone, string name) {
   return this.ClientScenesByName[zone][name];
public override void AfterEndInit() {
    foreach (StartSceneConfig startSceneConfig in this.GetAll().Values) {
       this.ProcessScenes.Add(startSceneConfig.Process, startSceneConfig);
       if (!this.ClientScenesByName.ContainsKey(startSceneConfig.Zone)) {
           this.ClientScenesByName.Add(startSceneConfig.Zone, new Dictionary<string, StartSceneConfig>());
       this.ClientScenesByName[startSceneConfig.Zone].Add(startSceneConfig.Name, startSceneConfig);
       switch (startSceneConfig.Type) {
       case SceneType.Realm:
           this.Realms.Add(startSceneConfig);
       case SceneType.Gate:
           this.Gates.Add(startSceneConfig.Zone, startSceneConfig);
       case SceneType.Match:
                                            // <<<<<< 自己加的
           break:
       case SceneType.Location:
           this.LocationConfig = startSceneConfig;
       case SceneType.Robot:
           this.Robots.Add(startSceneConfig);
           break:
       case SceneType.Router:
           this.Routers.Add(startSceneConfig);
           break:
       case SceneType.BenchmarkServer:
           this.BenchmarkServer = startSceneConfig;
           break;
       }
   }
}
```

### 2.9 HttpGetRouterResponse: 这个 ProtoBuf 的消息类型

- 框架里,有个专用的路由器管理器场景(服),对路由器,或说各种服的地址进行管理
- 主要是方便,一个路由器管理组件,来自顶向下地获取,各小区所有路由器地址的?想来当组件要拿地址时,每个小区分服都把自己的地址以消息的形式传回去的?

```
[Message(OuterMessage.HttpGetRouterResponse)]
[ProtoContract]
public partial class HttpGetRouterResponse: ProtoObject {
    [ProtoMember(1)]
    public List<string> Realms { get; set; }
    [ProtoMember(2)]
    public List<string> Routers { get; set; }
}
message HttpGetRouterResponse { // 这里, 是 Outer proto 里的消息定义
^^Irepeated string Realms = 1;
    ^^Irepeated string Routers = 2;
    ^Irepeated string Matchs = 3;// 这行是我需要添加,和生成消息的
}
```

### 2.10 HttpGetRouterHandler: IHttpHandler: 获取各路由器的地址

- •【匹配服】:因为我想拿这个服的地址,也需要这个帮助类里作相应的修改
- StartSceneConfigCategory.Instance: 不明白这个实例是存放在哪里,因为可以 proto 消息 进程间传递,那么可以试找,哪里调用这个帮助类拿东西?

• 这个模块: 现在还是理解不透。需要某个上午,把所有 RouterComponent 组件及其相关,再理一遍。

```
[HttpHandler(SceneType.RouterManager, "/get_router")]
public class HttpGetRouterHandler : IHttpHandler {
   public async ETTask Handle(Entity domain, HttpListenerContext context) {
       HttpGetRouterResponse response = new HttpGetRouterResponse();
       response.Realms = new List<string>();
       response.Matchs = new List<string>();// 匹配服链表 // <<<<<<<<
       response.Routers = new List<string>();
       // 是去 StartSceneConfigCategory 这里拿的: 因为它可以 proto 消息里、进程间传递, 这里还不是狠懂, 这个东西存放在哪里
       foreach (StartSceneConfig startSceneConfig in StartSceneConfigCategory.Instance.Realms) {
           response.Realms.Add(startSceneConfig.InnerIPOutPort.ToString());
       foreach (StartSceneConfig startSceneConfig in StartSceneConfigCategory.Instance.Matchs) {
           response.Matchs.Add(startSceneConfig.InnerIPOutPort.ToString());
       foreach (StartSceneConfig startSceneConfig in StartSceneConfigCategory.Instance.Routers) {
           response.Routers.Add($"{startSceneConfig.StartProcessConfig.OuterIP}:{startSceneConfig.OuterPort}");
       HttpHelper.Response(context, response);
       await ETTask.CompletedTask;
   }
}
```

### 2.11 HttpHandler 标签系:标签自带场景类型

```
public class HttpHandlerAttribute: BaseAttribute {
   public SceneType SceneType { get; }
   public string Path { get; }
   public HttpHandlerAttribute(SceneType sceneType, string path) {
      this.SceneType = sceneType;
      this.Path = path;
   }
}
```

# 2.12 LoginHelper: 登录服的获取地址的方式来获取匹配服的地址了。全框架只有这一个黄金案例

- 这个是用户登录前,还没能与网关服建立起任何关系,可能会不得不绕得复杂一点儿
- 这个地方,有点儿奇怪的是: RouterAddressComponent 这个组件,找不到添加的地方。它可能有点儿特殊,就是它不走常规 AddComponent<>()的方法添加,所以找不到添加,而只有获取。

```
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.RouterHtt
              await routerAddressComponent.Init():
              clientScene.AddComponent<NetClientComponent, AddressFamily>(routerAddressComponent.RouterManagerIPAddress.A
          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.A
          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);
            }
}
```

\*\*

### 3 服务器的功能概述:各服务器的作用(这个不是 ET7 版本的,以 前的)

- Manager: 连接客户端的外网和连接内部服务器的内网,对服务器进程进行管理,自动检测和启动服务器进程。加载有内网组件 NetInnerComponent,外网组件 NetOuterComponent,服务器进程管理组件。自动启动突然停止运行的服务器,保证此服务器管理的其它服务器崩溃后能及时自动启动运行。
- Realm: 对 Actor 消息进行管理(添加、移除、分发等),连接内网和外网,对内网服务器进程进行操作,随机分配 Gate 服务器地址。内网组件 NetInnerComponent,外网组件 NetOuterComponent,Gate 服务器随机分发组件。客户端登录时连接的第一个服务器,也可称为登录服务器。
- Gate: 对玩家进行管理,对 Actor 消息进行管理(添加、移除、分发等),连接内网和外网,对内网服务器进程进行操作,随机分配 Gate 服务器地址,对 Actor 消息进程进行管理,对玩家 ID 登录后的 Key 进行管理。加载有玩家管理组件 PlayerComponent,管理登陆时联网的 Key 组件 GateSessionKeyComponent。
- Location: 连接内网,服务器进程状态集中管理(Actor 消息 IP 管理服务器)。加载有内网组件 NetInnerComponent,服务器消息处理状态存储组件 LocationComponent。对客户端的登录信息进行验证和客户端登录后连接的服务器,登录后通过此服务器进行消息互动,也可称为验证服务器。
- Map: 连接内网, 对 ActorMessage 消息进行管理(添加、移除、分发等), 对场景内现在活动物体存储管理, 对内网服务器进程进行操作, 对 Actor 消息进程进行管理, 对 Actor 消息进行管理(添加、移除、分发等), 服务器帧率管理。服务器帧率管理组件 ServerFrameComponent。
- AllServer: 将以上服务器功能集中合并成一个服务器。另外增加 DB 连接组件 DBComponent
- Benchmark: 连接内网和测试服务器承受力。加载有内网组件 NetInnerComponent, 服务器 承受力测试组件 BenchmarkComponent。

### 4 Protobuf 相关,【Protobuf 里进程间传递的游戏数据相关信息: 两个思路】

- •【一、】查找 enum 可能可以用系统平台下的 protoc 来代为生成,效果差不多。只起现 Proto2CS.cs 编译的补充作用。
- •【二、】Card 类下的两个 enum 变量,在 ILRuntime 热更新库下,还是需要帮它连一下的。用 的是 HybridCLR
- •【三、】查找 protoc 命令下,如何 C# 索引 Unity 第三方库。

- •【Windows 下的 Protobuf 编译环境】:配置好,只是作为与 ET 框架的 Proto2CS.cs 所指挥的编译结果,作一个对比,两者应该效果是一样的,或是基本一样的,除了自定义里没有处理enum.
- Windows 下的命令行,就是用 protoc 来编译,可以参考如下. (这是.cs 源码下的)
  CommandRun(\$"protoc.exe", \$"--csharp\_out=\"./{outputPath}\" --proto\_path=\"{protoPath}\" {protoName}");
- 现在的问题是, Protobuf 消息里面居然是有 unity 第三方库的索引。
- 直接把 enum 生成的那三个.cs 类分别复制进双端,服务器端与客户端。包括 Card 类。那些编译错误会去天边。哈哈哈,除了一个 Card 的两个变量之外(CardSuits, CardWeight)。
- •【热更新库】: 现在剩下的问题,就成为,判定是用了哪个热更新的库,ILRuntime,还是 HybridCLR,如果帮它连那两个变量。好像接的是 HybridCLR. 这个库是我之前还不曾真正用 过的。
  - 相比于 ET6, 彻底剔除了 ILRuntime, 使得代码简洁了不少, 并且比较稳定

### 5 写在最后:反而是自己每天杳看一再更新的

- 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

### 6 现在的修改内容:

- •【Windows 下 proto2cs 消息转化】:上面拿匹配服地址,一个路由器相关消息加了参数,要重新生成一下。这个 mac 系统下不难,改天有时间也可以把 mac 下的运行环境配置好,这个功能模块就不用一定要切换电脑了。【手动做了一下】
- **【UILobbyComponent 可以测试】**: 这个大厅组件, Unity 里预设简单,可以试运行一下,看是否完全消除这个 UI 组件的报错,这个屏的控件能否显示出来?还是错出得早,这个屏就出不来已经报错了?
  - -【客户端】的逻辑是处理好了,编译全过后可以测试
  - -【服务端】:处理用户请求匹配房间的逻辑,仍在处理: C2G\_StartMatch\_ReqHandler. 必须记笔记。不记会忘。

- •【TractorRoomComponent】: 因为是多组件嵌套,可以合并多组件为同一个组件; 另早上看得一知半解的一个【ChildOf】标签,可以帮助组件套用吗? 再找找理解消化一下
- •【房间组件】: 几个现存的 working-on 的问题:
  - 多组件嵌套: 手工合并为一个组件。彻底理解确认后, 会合并
  - -【服务端】: 处理用户请求匹配房间的逻辑, 仍在处理
    - \*【数据库模块的整合】: 网关服在转发请求匹配时,验证会话框有效后,验证用户身份时,需要去【用户数据库】拿用户数据。ET7 留了个 DBManagerComponent, 还没能整合出这个模块
    - \*【匹配服地址】网关服的处理逻辑里,验证完用户合格后,为代为转发消息到匹配服,但需要拿匹配服的地址。ET7 重构里,还没能改出这部分。服务器系统配置初始化时,可以链表管理各小构匹配服,再去拿相关匹配服的地址。ET7 框架里的路由器系统,自己还没有弄懂。
- •【活宝妹坐等亲爱的表哥,领娶活宝妹回家!爱表哥,爱生活!!!】

### 7 TODO 其它的: 部分完成,或是待完成的大的功能版块,列举

- emacs 那天我弄了好久,把 C-; ISpell 原定绑定的功能解除,重新绑定为自己喜欢的 expandregion. 今天第二次再弄,看一下几分钟能够解决完问题?我的这个破烂记性呀。。。【爱表哥,爱生活!!! 任何时候,活宝妹就是一定要嫁给亲爱的表哥!!!】mingw64 lisp/textmode/flyspell.el键的重新绑定。这下记住了。还好,花得不是太久。有以前的笔记
  - Windows 10 平台下,C-; 是绑定到了 ISpell 下的某个功能,可是现在这个破 emacs 老报错,连查是绑定给哪个功能,过程报错都被阻止了。。。
- •【IStartSystem:】感觉还有点儿小问题。认为:我应该不需要同文件两份,一份复制到客户端 热更新域。我认为,全框架应该如其它接口类一样,只要一份就可以了。【晚点儿再检查一遍】
- 如果这个一时半会儿解决不好,就把重构的设计思路再理一理。同时尽量去改重构的 ET 框架 里的编译错误。
- •【Tractor】原 windows-form 项目,源码需要读懂,理解透彻,方便重构。
- 去把【拖拉机房间、斗地主房间组件的,玩家什么的一堆组件】弄明白
- •【任何时候、活宝妹就是一定要嫁给亲爱的表哥!!! 爱表哥、爱生活!!!】

### 8 拖拉机游戏:【重构 OOP/OOD 设计思路】

- 自己是学过,有这方面的意识,但并不是说,自己就懂得,就知道该如何狠好地设计这些类。 现在更多的是要受 ET 框架,以及参考游戏手牌设计的启发,来帮助自己一再梳理思路,该如 何设计它。
- ET7 重构里,各组件都该是自己设计重构原项目的类的设计的必要起点。可以根据这些来系统设计重构。【活宝妹就是一定要嫁给亲爱的表哥!!!】
- •【GamerComponent】玩家组件:是对一个房间里四个玩家的(及其在房间里的坐位位置)管理(分东南西北)。可以添加移除玩家。
- 【Gamer】: 每一个玩家
- •【拖拉机游戏房间】:
- •【爱表哥,爱生活!!! 活宝妹就是一定要嫁给亲爱的表哥! 爱表哥,爱生活!!!】