ET

deepwaterooo

February 19, 2023

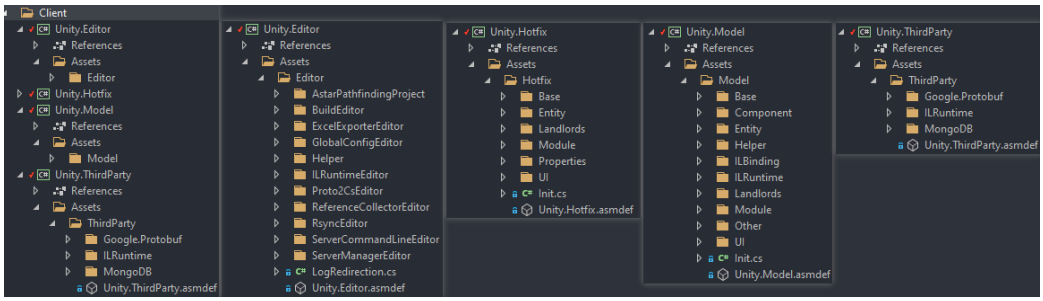
Contents

1 ET 框架框架设计模块功能等

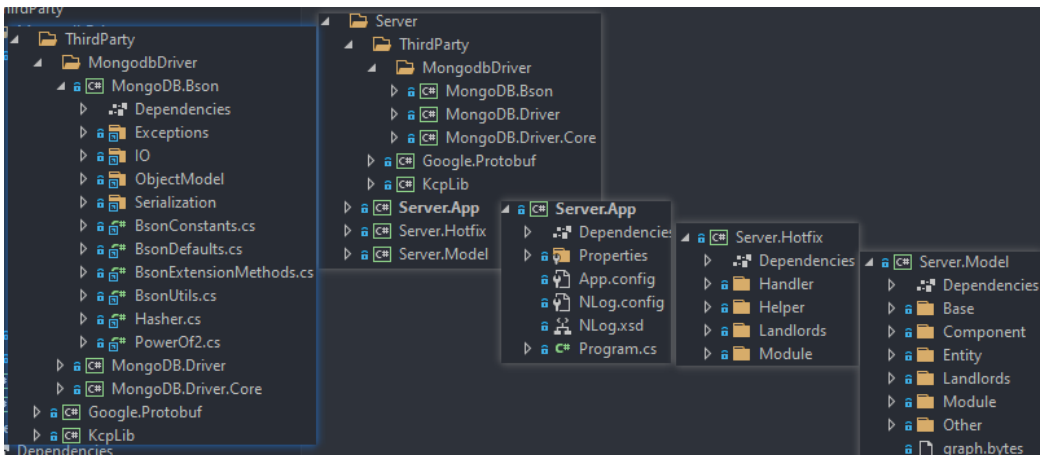
1.1 一个双端框架的 12 个项目，目录结构

- 就不知道这 12 个项目是如何组织构建起来的

1.1.1 客户端的四个项目



1.1.2 服务器端的 8 个项目：



- ET-EUI 整理出了一个小小的登录系统，如果这个系统能够运行得通，配置简单顺利，也就基本上也算是达到自己小小服务器的需要了。运行一下试试看。

1.2 模块功能管理

1.2.1 ET 框架中事件使用的规则和注意事项

- **事件结构体的定义**必须在 Model 或 ModelView 下，**事件的订阅和发布**必须在 Hotfix 或 HotfixView 下（是否为 View 层根据是否需要 UnityAPI 决定）
- 在 ET 框架中 **View 层可以直接调用逻辑层的代码**，而 **逻辑层不允许直接调用 View 层的代码**，所以逻辑层想要和 View 层交互只能使用抛出事件的方式，让 View 层进行订阅进行相应的处理。

1.2.2 ET 框架下 ETTask 的异步编程

- 开发早期都是使用协程或者多线程进行程序异步的实现，
- 在 C#5.0 之后引入了 Tasync await 等关键字可以轻松优美的实现程序的异步进行。
- Task 是 C# 异步编程库的异步功能基础类型，包含了多线程的使用。
- 而 ET 框架主打的是 **多进程单线程**，所以使用 ETTask 对 Task 进行了封装，使其不用考虑多线程的共享难题，更易于使用。

1.2.3 协程

- 协程其实就是创建一段程序辅助主线程的运行，注意 **协程不是多线程，其仍运行在主线程当中，其只是将一个函数按照一定的条件分解成若干块，穿插在主线程的各个位置中运行**。
- async 和 await 关键字
  - async 是修饰函数的关键字，被修饰的函数称为异步函数，只有被此关键字修饰的函数才能使用 await 关键字
  - await 关键字后跟一些表达式 (一般是返回值为 ETTask 的函数)，在异步函数运行到此时会立即跳出函数，继续执行原逻辑。
  - await 返回前会做出保证，保证 await 后表达式的任务完成后，执行点会跳回异步函数中，继续向后执行剩余代码
  - 若 await 后表达式正常返回，可用变量接收表达式的返回值，表达式的返回值类型为定义表达式 ETTask<> 的泛型

2 小小服务器：要怎么才能开始动手试图去实现这个小服务器呢？

2.1 如果适配 ET 框架，现游戏可能哪些模块版块存在问题

- 我也觉得 ET 框架可能不太适合我现在的游戏（也就是说，把我的游戏完全适配成 ET 框架来开发，原本只需要一个小小服务器，完全适配为 ET 框架，就把问题弄得狠复杂了。。。），
- 使用 ET 框架，我所有的安卓基础就会被抛到九宵去外，不再关安卓 SDK NDK 什么事儿了。。。。。是对自己太大的损耗。而我原本还可以简单封装实现的安卓录屏，游戏内使用安卓 SDK 相关功能模块录屏游戏过程等，会被全部废掉，损失太大不值得。我觉得我就只要个文件服务器加个数据库而已。
- 原因是：我现在还想不通若是用 ET 框架来实现自己游戏的（服务器与客户端双端都可以热更新），我该如何实现我的方块砖 10 个按钮上的点击事件，射线检测？它的 ILRuntime 热更新程序域里对射线检测包的组件安装可能会成为自己狠大的问题，因为还不是狠懂里面的内部原理. 这个模块重构的原理是: 把射线检测, 如果必要一定要, 封装成如 ET 中任何其它可装载卸载的组件一样的装载到某个必要场景上去.
  - ET 里有个检测鼠标左右键位置的帮助类, 但跟射线检测应该还是相差狠远的. 而游戏场景里面有一个 OperaCompoent，这个组件会实时监听按键的点击并且将点击的位置发送给服务器，服务器再传送回客户端
- 所以, 现在对这个框架, 最深的感触是: 盲人摸象, 摸每部分细节似乎都快清楚了, 却无法组装从一个更高的层面上来理解和把握框架设计, 无法吃透, 在大框架功能模块上犯难, 在网上再搜一搜
- 我可以把两组按钮画面同样做成预设与热更新资源包，射线检测同样会成为可装载可卸载的组件，可是再后面射线检测到的物体逻辑，感觉有点儿复杂了
-

2.2 如果不适配，怎么弄个服务器带数据库等逻辑？

- 使用部分共享源码的双端（共享的是文件服务器 8 个项目，MongoDB 数据库服务器，Realm 注册登录用，网关服，Location 服，ETTAsk，RPC 消息机制，NetComponent 等自己机对陌生需要练习，而自己的服务器也不可缺省的相关功能）

- 现在知道自己整的不沦不类的服务器所谓的登录，登录的是网页上的认证相关，跟自己真正要实现的游戏里注册登录服保存数据库完全两回事，现在知道了。
- 作用 ET 的头，实现用户注册与登录，适配自己现有游戏的尾，游戏除了入口之外全游戏进热更新程序域里
- 那么自己的现框架架构作尾，全游戏逻辑进热更新域，存在的问题就变成是：
- 我无法再实时动态检查用户上下线顶号之类的，我只能默认登录过就是登录状态，可是用户下线了，或更严格的说掉线了，服务器并不及时知道，可以通过安卓 SDK 中的按钮登出知道。但是掉网了掉线了呢？（这部分的逻辑可以晚点儿再考虑，把网络请求相关的摸得再熟悉一点儿再弄）
- 再则，ILRuntime 热更新程序域里，我又该如何实现在热更新程序域里网络上载用户的游戏保存进展？这里需要去想和理解，为什么它 ET 框架就可以在热更新程序域里同网络交互，你哪里还没有想明白？
- ET 框架，热更新程序域里装载的组件，只是帮助与外界游戏程序域连通好，真正的网络请求上传下载等是在热更新域外面完成链接式传进去的？感觉对这个大框架没有掌握好，脑袋仍然是在像糊糊一样。。。
- ET 框架，网络的那部分做得还是比较完整的。实现在了各种的封装，涉及大量的网络调用与交互，游戏过程中的交互与更新。但是太多的功能对于自己的游戏来说完全不必要. 所以只想用 ET 的头
- 各种泛型，接口的定义，一二三个参数等的泛型接口定义 (你可以去找一找工程中的各种 ILRuntime 的适配器)，全都是都可以成为热更新域里能够被游戏程序域里识别的原因，所以狠多设计，自带 ILRuntime 的适配性质
- 那么就可以小一点儿一点儿地来, 先弄个登录窗口, 实现服务器的注册登录保存登录信息到数据库, 相对比较小点儿的逻辑. 这个过程中把 MongoDB 数据库的配置等所有连接过程中必要的步骤, 可能出现的问题给解决掉, 就算前进了一小步呀
- 不知道怎么开始，也不知道怎么创建可以🄵套的像是安卓模块库一样的子工程，就只能把小游戏斗地主复制一份了再从它的基础上来改?!!!
- 如果简单一点儿开始，我觉得我应该是可以先把简单点儿的 MongoDB 数据库连接成功，把用户登录相关的逻辑，网络交互的部分，ETTask RPC ACTOR 消息等，哪怕是复制，把这部分弄过去

2.3 ET 框架

- https://blog.csdn.net/qq_33574890/article/details/128244264?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EAD_ESQUERY%7Eyljh-1-128244264-blog-123841252.pc_relevant_multi_platform_whitelistv4&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EAD_ESQUERY%7Eyljh-1-128244264-blog-123841252.pc_relevant_multi_platform_whitelistv4&utm_relevant_index=2 上次看看得不是狠懂，这次再看，至少是觉得 UI 的逻辑处理，作者的观点更自然真实一些，放在一个文件一起处理，个人认为更好, 而不是折分成为几个文件

```
class LoginState:State{
^^Ivoid OnEnter(){
^^I^^IUI.Show()
^^I}
^^Ivoid OnLeave(){
^^I^^IUI.Hide()
^^I}
}
```

3 登录协议流程

- 因为登录协议是客户端与服务器通信的，不属于服务器内部协议，所以打开 OuterMessage.proto，里面存放的都是客户端与服务器通信定义的协议数据。
- 比如定义如下，登录协议：

```
message C2G_LoginGate // IRequest
{
^^Iint32 RpcId = 90;
^^Iint64 Key = 1;^^I// 帐号
}

message G2C_LoginGate // IResponse
{
^^Iint32 RpcId = 90;
^^Iint32 Error = 91;
^^Istring Message = 92;
^^Iint64 PlayerId = 1;
}
```

- emacs 里 org-mode exporte-to-pdf 希望有个 latex 选择可以自动将^I 转化为空格，而不是这种字符，晚点儿再弄这个
- 注意点：**没有意识到像是注释一样的片段，这个协议里，会成为标注或是标签**
  - 1. 因为登录是请求-响应类型协议（即发送一条数据，并期望返回一条数据），所以注意对应 C2R_Login 协议带有 “//ResponseType R2C_Login” 标志，在生成协议时，用于标记这个 C2R_Login 请求对应的响应类型为 R2C_Login
  - 2. 因为请求是直接发送给 realm 服的，所以是普通的 IRequest 类型协议，标记为 IRequest
  - 3.R2C_Login 回复类消息结构，因为是 Realm 服发送给客户端的，因此是一个普通 IResponse
  - 4. 注意两个协议类里面都有 RpcId，主要用于发送请求-响应类消息时，发送将自己的 RpcID 发送出去，返回时带回这个值，用于发送方接受到返回协议时，可以找到对应的是哪一个请求协议返回来的。

4 一步一步的进展

- 现在还是不想这么改，因为项目太大了，搞不清哪是哪里，另外狠多类是部分类，就是一个类可以有好几个地方来定义扩展这个类，然后同样好几个同样的类，就不知道该在哪个类里定义。12 个项目之间的交互有点儿复杂，现在没有吃透，只适合小范围改动去帮助自己理解项目，还不适合自己从头到底来重构自己的项目。就按网络上的建议，先把注册登录系统的逻辑再修改一遍，直接再接自己的热更新程序域
- 再往下走。因为接自己项目的入口就全进自己项目的热更新域仍然存在诸多的登录登出掉线，以及用户游戏进展数据的保存等相关逻辑。可是因为急需要下手练习，顾不上这么多，先练再说，练到哪里是哪里。。。。。
- 首先，把斗地方大厅改写为游戏主菜单的三个选项 (如果我只想用 ET 的头，它的头太大了，还是要自己弄个小小的头，小小的服务器，所以暂时就还是考虑自己从头实现一个 MongoDB 的小小服务器比较容易一点儿，不懂的就翻 ET)

- 把这个界面的相关上下文全部适配好：UI 的自动创建生成系统，UI 的按钮点击回调等
- 这里想要找的是：在点击的回调里如何，是否可以卸载装载 UI 组件，还是说必须得去 HotfixView 什么视图层来处理这些逻辑呢?

4.1 UIType.cs: 这种类型的定义好像不止加一个地方，一个地方不够，可是大的框架架构还是没搞明白

```
namespace ETHotfix {

    public static partial class UIType {
        public const string Root = "Root";
        public const string UILogin = "UILogin"; // 注册 登录 界面
        public const string UILobby = "UILobby"; // 主菜单  三选项

// 上面的界面远远不够呀。。。
        public const string UIEducationalMode = "UIEducationalMode";

        public const string UIEducational = "UIEducational";
 // 怎么再把它细化为：三  四  五方格呢？应该是要用同一接口的不同实现，完全重复写三个系统会把人弄死的。。。。
        public const string UIGridThree = "UIGridThree";
        public const string UIGridFour = "UIGridFour";
        public const string UIGridFive = "UIGridFive";

        // 那么就涉及游戏界面的折分：哪些是可以公用，哪些是不得不细化最小粒度的？

        public const string UIClassic = "UIClassic";

        public const string UIChallenge = "UIChallenge";
 // 挑战难度：要定义接口来实现 20-50 个不同的实现了？
    }
}
```

- 安卓 SDK 这个框架其实并不受影响。但本质是所有安卓 SDK 的东西不能够热更新。因为 ET

是网络多人游戏框架的，可能更多的是不适合添加与适配案桌 SDK。这些晚点儿再得结论好了，反正我的案桌 SDK 本质也是可要可不要。如果能够快速掌握一个比较好的双端框架的话

- 不知道若是照这么改下去，得把这个游戏改成是什么花葫芦呢?

5 带 MongoDB 数据库的注册登录用户帐户管理资源文件服务器

- 去找和实现简单的服务器项目，操纵 MongoDB 数据库
- 除了自己的电脑安装有 MongoDB 数据库之外，服务器项目中因为要连接操纵电脑上数据库，可能还需要狠多插件的安装与配置，连接字符串，什么 MongoDBClient 之类的。这些细节就只能找到一个小参考项目，自己试着连接，出错了再一一更正自己可能存在的错误，要真正能够把项目连通运行得起来，才算真的解决问题
-

6 ET 框架-19 ET 框架账号中心服逻辑编写（1）

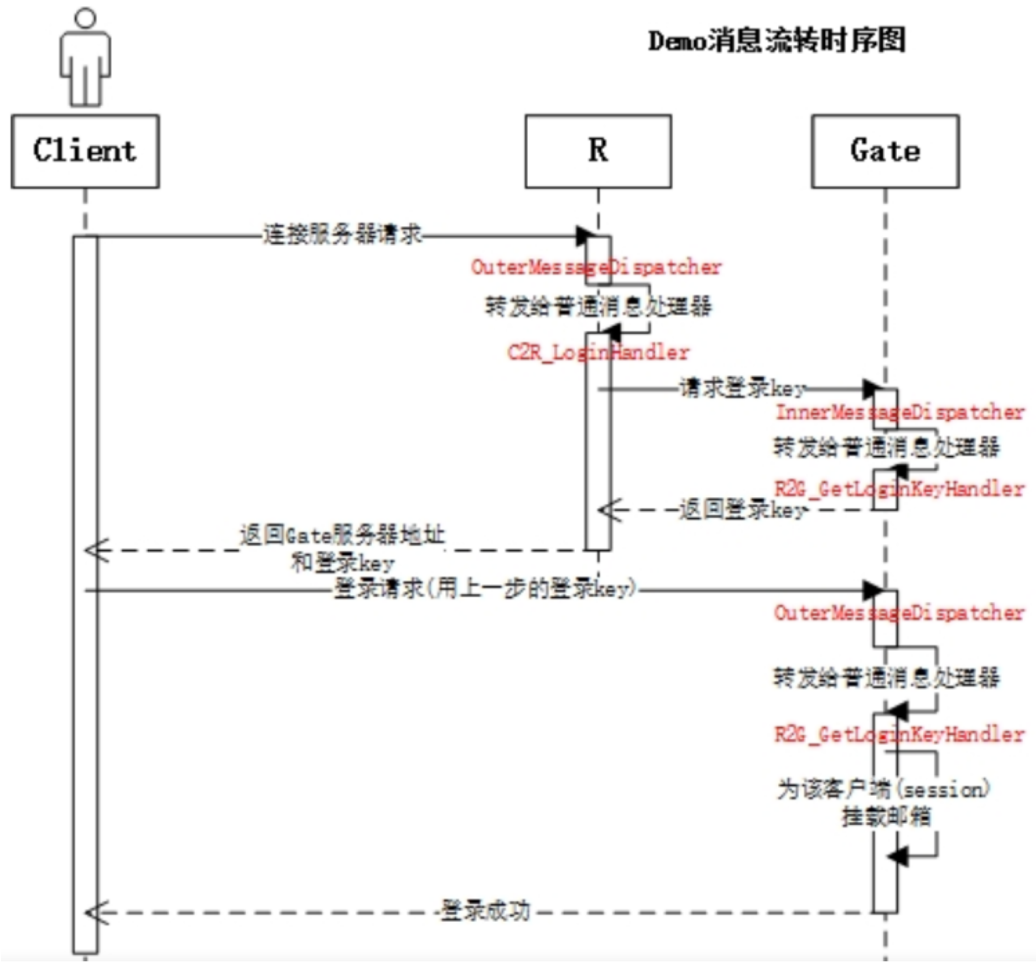
- https://blog.csdn.net/m0_48781656/article/details/124899665?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-124899665-blog-123592622.pc_relevant_multi_platform_whitelistv3&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1pc_relevant_multi_platform_whitelistv3&utm_relevant_index=2
- 现在，对 ET 框架里，热更新层，不变层的大的框架层次，稍微开始有点儿概念。还需要再多理一理。现在觉得，只要自己感兴趣，想要找的逻辑都能够找得出来。只是还需要多运行几遍帮助理解消化。
- 但是现在什么都有了，应该能够很好地实现 ET 的头加自己项目的主体部分了。客户端的逻辑基本都懂了，服务器端需要明天上午再看一下
- 只是仍需要考虑，将来如何自己写个多人小游戏?

7 ET 框架登录流程

- 1. 服务器由于在启动时就添加了 NetOuterComponent 组件，默认状态下使用 Tcp 协议，组件启动时，指定了自己的消息解析器为 ProtobufPacker，消息派发器为 OuterMessageDispatcher。
- 2. 由于 NetOuterComponent 继承了 NetworkComponent，然后在 awake 里，调用了 NetworkComponent 的 Awake 方法。
- 3. 实例化一个 TService，并监听启动端口，绑定 OnAccept。
- 4. 当客户端的协议通过端口与 IP 发送 C2R_Login 协议时，TService 监听到一个新的连接，在底层建立一个 TChannel（内部封装了 socket）与之关联，回调 OnAccept。
- 5.OnAccept 回调新建一个 Session，并将建立的 TChannel 与这个 Session 关联起来，并启动 Session，内部会调用 TChannel 的 Socket 启动循环获取数据。
- 6. 底层 Socket 获取数据会通知 TChannel 进行处理，进而将处理好的流数据回给 Session 的 OnRead 进行序列化，协议号解析等操作。

- 7. 消息类型为 IRequest，所以直接交给 OuterMessageDispatcher 协议消息转发器进行处理。
- 8.OuterMessageDispatcher 发现是非 Actor 消息，调用 MessageDispatcherComponent 组件直接对消息进行处理。
- 9. 由于 C2R_LoginHandler 类之前就已经在 MessageDispatcherComponent 注册好，所以 C2R_Login 消息直接交给他处理。
- 10. 首先应该根据数据库进行帐号密码验证（这一步 ET 在 DEMO 中省略了），然后根据配置，拿到一个随机的 Gate 服务的端口与地址，通过内网组件 NetInnerComponent 拿到一个通向该地址与端口的 Session。然后 Reaml 模块向 Gate 模块发送一个 R2G_GetLoginKey，为客户端请求一个唯一 Key 当作连接 GATE 时的鉴权，这里使用了异步 await。**注：在 ALLSERVER 模式下，没有区分 Reaml 模块与 Gate 模块，但实际流程是一样的**。
- 11. 身为 Gate 服务模块的 NetInnerComponent 会监听自身提供的端口，同样的 Session 会将消息解析好之后发给 NetInnerComponent 的 MessageDispatcher（即 InnerMessageDispatcher）进行处理。
- 12.InnerMessageDispatcher 发现 R2G_GetLoginKey 是一个普通 IRequest 消息，因此直接交给 MessageDispatcherComponent 的 Handle 进行处理。
- 13. 同样的 R2G_GetLoginKeyHandler 类也注册好了，直接到 R2G_GetLoginKeyHandler 的 Handle 方法中，由于 R2G_GetLoginKeyHandler 继承自 AMRpcHandler，在 AMRpcHandler 中，构造了一个 Reply 函数传给 Run 方法，同时也将 Response 实例化 (这里是 G2R_GetLoginKey 类实例) 传给 Run。
- 14.R2G_GetLoginKeyHandler 实现了 Run 方法，Run 方法处理 R2G_GetLoginKey。他会随机一个 64 位的数放入到 G2R_GetLoginKey 的返回数据中，并调用 Reply。
- 14.Reply 内部会调用 NetInnerComponent 一路传过来的 Session 进行 Session 的 Reply 方法调用，直接将 G2R_GetLoginKey 发送回去。
- 15.Reaml 与 Gate 连接的 Session 收到了数据，由于是 G2R_GetLoginKey 是 IResponse 类协议，所以不走 MessageDispatcher 转发了，直接走之前发送时注册的回调函数处理
- 16. 回调函数内部是直接将异步 TCS 设置 SetResult，来唤醒回调，这样一步步向上唤醒最终回到第 10 步中的 await 处。
- 17. 可以看到 C2R_LoginHandler 也是继承自 AMRpcHandler，所以再 AMRpcHandler 内也封装了一个 reply 回调，以及一个 R2C_Login 实例传给 C2R_LoginHandler 的 Run，现在回到 await 处。
- 18. 通过配置拿到 Gate 服务模块的外网 OuterConfig 的 Address2，以及上面回来的 Key，传给 response，调用 reply 通过 session 将消息发出去。
- 19. 同样的方式，客户端收到一个 IResponse，调用之前 Call 发送数据时注册好的回调，回调内部设置 tcs.SetResult，唤醒异步方法。从而回到客户端发送请求的 await 处。
- 20. 客户端拿到请求后，通过发来的 Address 新建一个与 Gate 的连接，然后发送 C2G_LoginGate
- 21.Gate 服务收到协议后，通过一系列底层转发，周转到 C2G_LoginGateHandler 类进行处理。

- 22. 验证成功后，Gate 为每个玩家新建一个 Player 实体与之对应，并交给 PlayerComponent 进行管理，同时将与之关联的 Session 上挂载 SessionPlayerComponent，将实体 Player 与关联的 Session 关联起来，最后为关联的 Session 绑定一个 MailBoxComponent。这样 Gate 的内网组件就能将 Actor 消息转到这个 Session 进行处理时，可以直接当 entity 为这个 session 进行转发即可，注意 Player 基类为 ComponentWithId，所以 Player 的 ID 即为他的 InstanceId。同时将这个 ID 发回给客户端，当作唯一 ID 来使用（猜测是在 Gate 上的实体，不会发生转移的原因）。
- 自此，客户端已经完整的登录到 Gate，后续客户端的通信全部与 GATE 进行通信，Gate 对客户端协议进行转发，收到其他服务传过来的 Actor 消息，也能转发到对应的客户端 Session 进行处理。
- 用一个图来表达的话，就成为下面的样子：也是网友总结的：

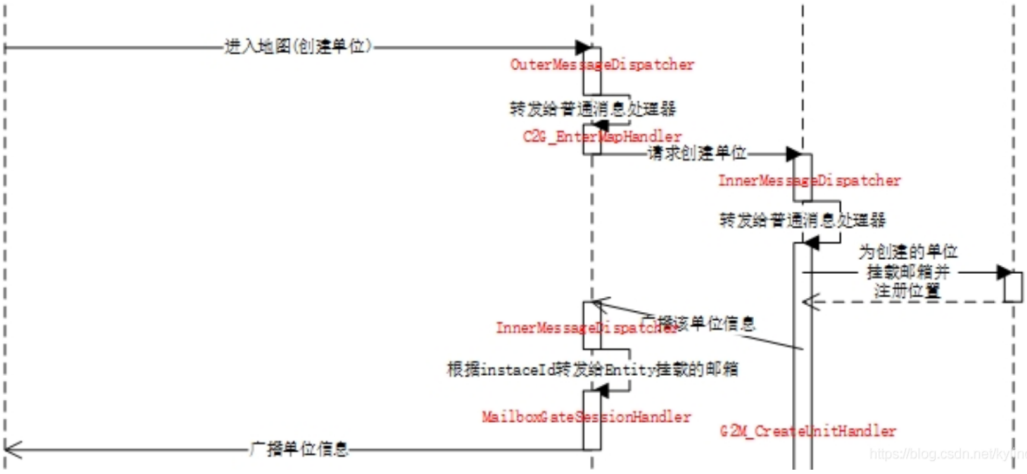


8 与其他服务模块通信基础（以 Map 服为例）

- 客户端想要进入其他服务模块，首先都会再该服务上创建或者移动一个实体进去。
- 这里以与 Map 服通信为例。

- 1. 客户端向 Gate 发送一条 C2G_EnterMap 协议，GATE 收到服务的细节这里就不多说一遍了，由于 C2G_EnterMap 是 IRequest，所以最终由 C2G_EnterMapHandler 类进行处理。
- 2. 通过与之关联的 Session 上 SessionPlayerComponent，能拿到在 Gate 上的 Player 信息。然后再通过配置拿到一个 Map 服务的地址，通过 NetInnerComponent 在 Gate 上构建一个与选好的 Map 服务进行通信的 Session。
- 3. 向这个 Session 发送一条 G2M_CreateUnit 协议，将 Gate 上的 player 实体 ID，session 的 InstanceId 都发送过去。
- 4. 经过底层处理后，由 G2M_CreateUnitHandler 类处理，首先创建一个 Unit 实体，**与在 Gate 上创建 Player 实体不同，Player 的 ID 赋值为组件的 InstanceID，而在 Map 上的 Unit 实体，通过 IdGenerater.GenerateId() 方法生产一个唯一 ID 作为实体的 Id**。
- 5. 给 Unit 上添加移动，寻路组件，并给一个初始化的位置，并为其添加一个 MailBoxComponent 代表这个 unit 为一个 Actor。由于在 Map 上的 Unit 会发生转移到其他服务模块上的可能，所以需要通过 MailBoxComponent 的 AddLocation 方法，向 Location 服务模块，注册自己的唯一 ID，与自己的 InstanceId。
- 6. 通过 NetInnerComponent 获取一个与 Location 服务连接的 Session，并发送一个 ObjectAddRequest 协议，经过一系列处理后，在 Location 服务上由 ObjectAddRequestHandler 类处理，调用 LocationComponent 的 Add 将 ID 与 InstanceId 注册好，然后直接返回。
- 7.Map 服务上收到 Respose,Session 唤醒异步，回到第 5 步，注册好定位服务（Location）后，给 Map 上的 Unit 添加一个 UnitGateComponent 组件，将 Gate 发过来的 Gate 与客户端连接的 Session 实例 ID 给保存到 Unity 上。
- 8. 将这个 Unit 放入到 UnitComponent 组件中进行管理。将生成的唯一 Unit ID 添加到回复的 response 协议中。
- 9. 创建一条 M2C_CreateUnits 协议，查询 UnitComponent 组件，遍历所有已经存在的 Unit，将数据添加到 createUnits 协议数据中，广播 M2C_CreateUnits 协议。
- 10.M2C_CreateUnits 协议类型是 IActorMessage，进入 MessageHelper，获取所有的 Unit，获取到 ActorMessageSenderComponent 组件。
- 11. 遍历所有 unit，获取 unit 上 UnitGateComponent 组件获取连接状态，通过 actorLocationSenderComponent 以及每个 Unit 上挂载的 unitGateComponent 的 GateSessionActorId，这个值是创建 Unit 时，存储的从 GATE 上与客户端连接的 Session 的 InstanceId。
- 12. 因为在 GATE 上的实体不会发生转移，所以他的 InstanceId 很稳定（个人觉得也可以拿 GATE 上 Player 的 InstanceId 做后续处理），通过 InstanceId 即可拿到对应生产他的 Gate 的端口与地址，通过端口与地址，新建一个 ActorMessageSender 实例。
- 13. 通过 actorMessageSender 发送上面的广播协议，通知所有客户端新的实体被增加了。这里实际走的是向 Gate 发送了一条 IActorMessage 协议，然后 Gate 内网组件收到，转给 InnerMessageDispatcher 处理，再转给 mailBoxComponent 处理，然后调用到 MailboxGateSessionHandler，将协议转发给客户端即可。**备注：这里有个小处理：iActorMessage.ActorId = 0，不暴露内部参数**
- 14. 广播通知所有客户端生成了一个新的 Unit 后，回复一个 M2G_CreateUnit。
- 15. 再次经过一系列处理后，Gate 服务收到 M2G_CreateUnit，依然是由 Session 唤醒异步到 Gate 的 C2G_EnterMapHandler 的创建请求处。
- 16. 将创建好的 UnitId 即 Map 上的 Unit 这个唯一 ID（注：与 InstanceId 不同，他是在 Location 中绑定过的），绑定到 Gate 上 Player 的 UnitId 上，同时赋值给 G2C_EnterMap 协议中，发回给客户端。这样后面 Gate 进行双向转发时，都能通过这个 UnitId 来进行。

- 自此，创建好 Map 上 Unit，绑定了 UnitId，在 Location 上注册了 Map 上的 Unit，便于发送 ActorLocation 协议

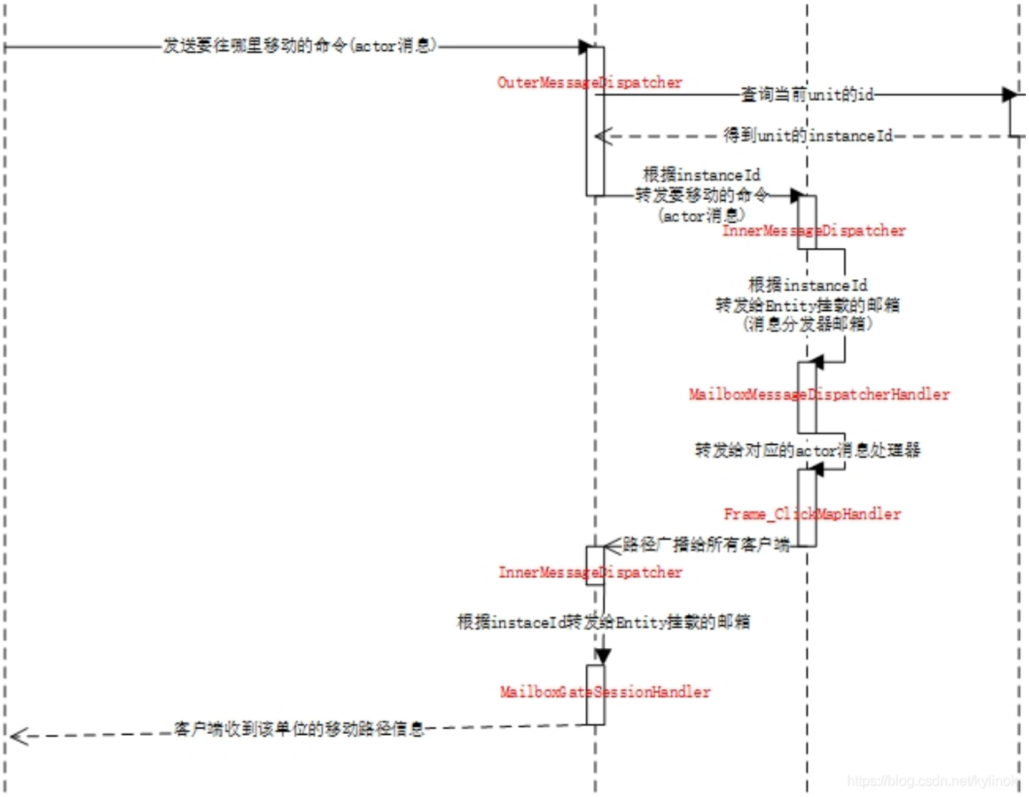


9 与其他服务模块通信 Actor（以 Map 服为例）

- 现在客户端已经保有 Map 上 Unit 的唯一 ID，Gate 上 Player 的实例 ID，同时 Gate 上也保有这两者，而 Map 上的 Unit 身上保有 Unit 的唯一 ID，还有 Gate 与客户端连接的 Session 的实例 ID。
  - 1. 客户端发送 C2M_TestActorRequest 协议，Gate 收到此消息，中转到 OuterMessageDispatcher 派发器的 actorLocationRequest 消息处理。
  - 2. 通过 ActorLocationSenderComponent 以 Player 的 unitId 为 Key 拿到一个 ActorLocationSender，内部包含访问 Location 服务器拿到 unitId 对应的最新 InstanceID 等操作。
  - 3. 通过 actorLocationSender 转发 C2M_TestActorRequest 协议给拥有的 UnitID 对应实体 Unit 的 Map 服务。
  - 4.Map 服务通过内网组件，一路转到 InnerMessageDispatcher 上，然后通过拿到的 Unit 的 InstanceID, 找到对应的 Entity，获取他身上的 MailBoxComponent，然后一步步中转到 MailboxMessageDispatcherHandler 进行处理。
  - 4. 获取 ActorMessageDispatcherComponent 处理,最终交由具体的处理类 C2M_TestActorRequ 进行处理，这里填充好协议内容后，直接返回协议，交由连接的 Session 发送数据。
  - 5. 然后又经过一层层处理,由 Gate 上的连接到 Map 上的 Session 收到 M2C_TestActorResponse 由于它是一个 Response，所以走回调，唤醒异步，回到 Gate 的 OuterMessageDispatcher 中的 actorLocationRequest 处理，得到回复的消息：IResponse response = await actorLocationSender.Call(actorLocationRequest);，Actor 消息已经转发成功并拿到回复了。
  - 6. 由 Gate 的 OuterMessageDispatcher 直接调用与客户端连接的 Session 进行 Reply，将由 Map 发来的 M2C_TestActorResponse 数据发给客户端。
  - 7. 客户端收到 M2C_TestActorResponse，老一套，它是一个 Response，唤醒异步，回到客户端，发送 C2M_TestActorRequest 的地方：M2C_TestActorResponse response = (M2C_TestActorResponse)await SessionComponent.Instance.Session.Call(new C2M_TestAc { Info = "actor rpc request" });

- 自此，一整个完整的 Actor 消息就流转完毕了。

下图是客户端派发一个移动的 Actor，与上面的测试 Actor 差不多。



- 爱亲爱的表哥，活宝妹一定要嫁的亲爱的表哥!!!，爱生活。活宝妹一定要嫁给亲爱的表哥。爱亲爱的表哥，活宝妹一定要嫁的亲爱的表哥!!! 爱生活!!!