三国网络设计

**网络涉及模块：**

Client：

网络游戏客户端，主动向服务器发起连接。

Gated：

认证服务器，ip+port对外公开，主要用于身份认证，通过后根据当前服务器负载会分配合适的Transd给Client。

Transd：

连接服务器，可能有多台，ip+port从Gated处获取，也起到应用层防火墙的作用。

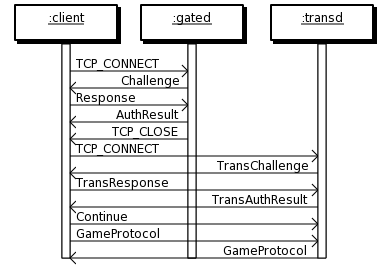
Statusd：

游戏服务器状态异常或者维护时，Client通过Statusd获取游戏服务器状态信息。

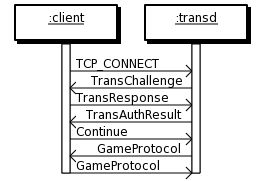
Loadd/Playerd/…：均为内部服务器，对客户端不可见，暂不描述。

**客户端和服务器间协议交互流程：**

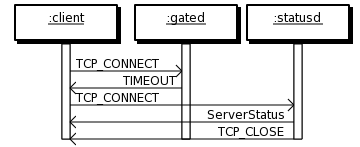
首次连接：



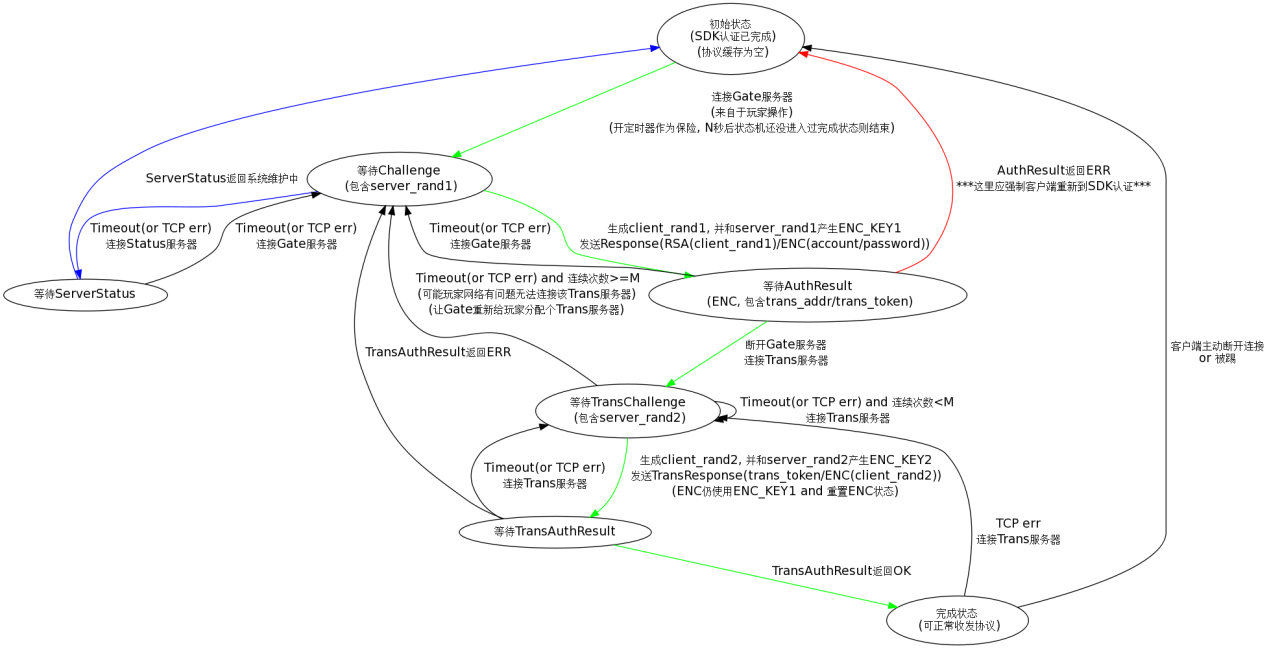
意外断线后的重连：



无法连接到游戏服务器时：



状态迁移图：



**协议说明**：

玩家认证和通信密钥协商：

Challenge：

Client连接到Gated之后由Gated主动发送，提供一个随机串server\_rand1用于生成密钥。

Response：

Client将account/token等信息加密后发送给Gated，其密钥key1来自于server\_rand1和client\_rand1（Client自行产生的随机串，是整个通信过程的安全核心），并将client\_rand1用Gated的公钥加密并发给Gated。

AuthResult：

Gated验证account/token信息，通过则分配一个Transd给Client，该协议包含了代表Client的trans\_token以及Transd的ip/port。

TransChallenge：

Client连接到Transd之后由Transd主动发送，提供随机串server\_rand2用于生成密钥。

TransResponse：

Client提供trans\_token用于表明身份，并使用key1和随机串salt加密client\_rand2（随机串，用于后续的加密通信）和server\_rand2(用于Client认证)，并将其与salt一起发送给Transd。

salt的主要作用是保护key1。

client\_rand2其实有点冗余。

TransAuthResult：

Transd验证trans\_token和Client认证信息。通过后返回结果给Client。

该协议之后开始对通信数据加密。

断线重连：

TransResponse：

Client通知服务器Client当前分别已发送和已接收的GameProtocol数量，用于让服务器判断是否可恢复连接。

device\_id用来标志Client当前设备，如果发生了设备更换，则无法自动恢复连接。

TransAuthResult：

Transd通知Client服务器当前分别已发送和接收的GameProtocol数量，用于让Client判断是否可恢复连接。

do\_reset字段提示Client是否需要完整执行游戏流程（比如从获取角色开始），可能原因是首次登陆，或者底层网络断线期间大量数据包丢失导致无法恢复。

Continue：

Client通知Transd是否恢复或者重置连接。如果选择恢复，则在此协议发送之后，Transd和Client分别开始重发断线期间丢失的GameProtocol。如果选择重置，则服务器和Client各自将GameProtocol计数归零，重新开始。

Continue之后（包括重传），Client和Transd才能各自分别开始发送GameProtocol。

游戏逻辑：

GameProtocol：

应用层协议，具体含义根据游戏内容而定，处理在lua部分。

其他：

Kickout：

提示Client强制玩家下线。

ServerStatus：

服务器状态信息。

**客户端如何使用网络模块：**

1. 实现IGameClient接口。
2. 初始化Connnection，并连接服务器：

样例代码参考Program.cs中的Main部分。

使用Connection::IsEstablished()查询是否连接已建立。

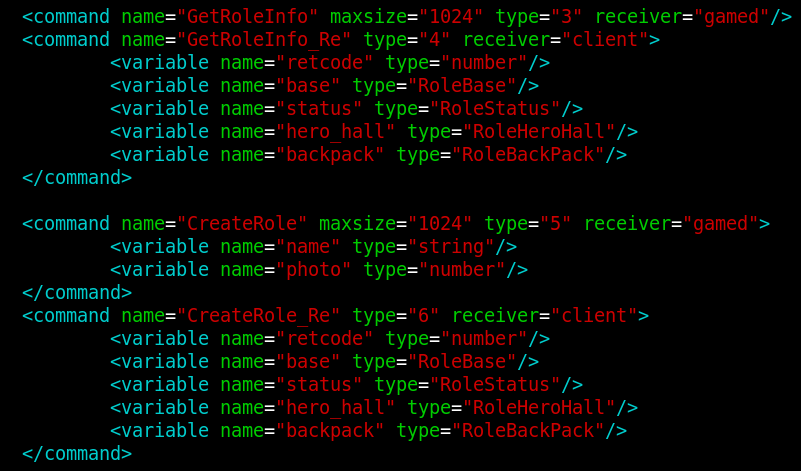
3，发送和接收应用协议。

**使用应用协议：**

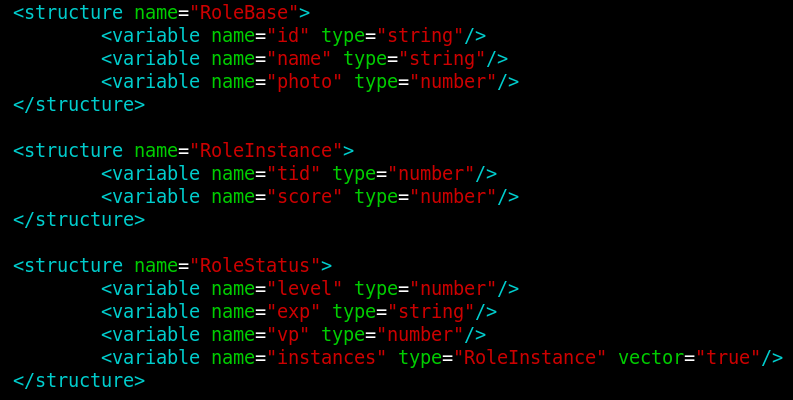
应用协议即游戏相关协议，由程序员（一般是服务器程序）根据游戏功能来定义，并将协议定义文件（xml）和最终生成的lua协议代码分别同步到客户端和服务器。客户端程序需要将所有lua协议代码load入对应lua\_state。

客户端除cmds目录下的lua代码外，其他代码都是自动生成，不要修改。

应用协议定义文件样例：

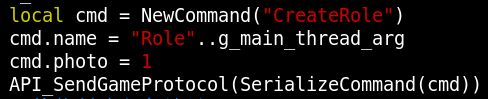


该文件分别定义了4个协议，用于获取角色信息（请求+回应协议），创建角色（请求+回应协议）。其中的RoleBase/RoleStatus等都是结构体，定义如下：



使用lua发送应用协议：

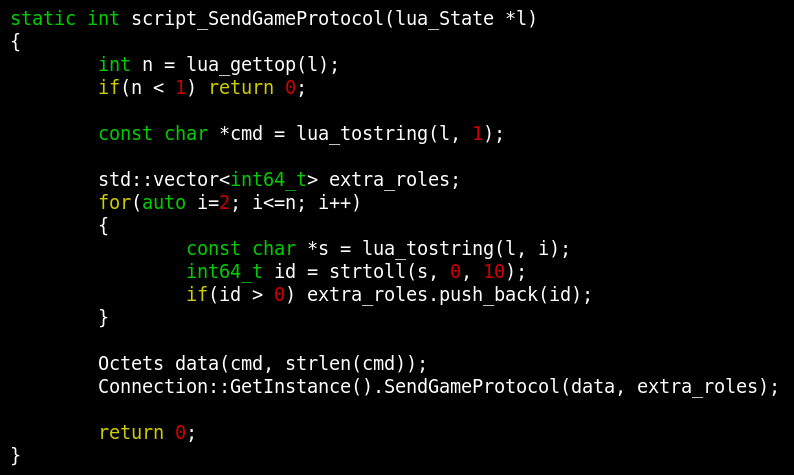
以创建角色为例：



其中NewCommand/SerializeCommand为框架函数；

CreateRole为协议名，name，photo为协议成员变量。

API\_SendGameProtocol用来发送应用协议给服务器，改函数需要客户端程序自行实现，最终该函数会调用Connection::SendGameProtocol实现，一个C++的例子。



使用lua处理应用协议：

协议处理已经Lua框架化，但需要客户端程序修改TransClient.cs，将接收到的data作为参数传给lua的DeserializeAndProcessCommand函数。

最终的协议处理代码类似下图（CreateRole.lua，这是服务器处理协议的代码，仅作示例）：

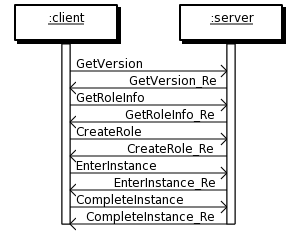


Player，role，others对客户端无定义，即DeserializeAndProcessCommand的第1和2以及4+参数，客户端程序可根据自身需求决定其意义。

arg为协议参数，包含了CreateRole\_Re定义的成员变量，比如retcode,name,level等，具体内容参看cmds.xml。

others只有服务器才有，不用管。

**游戏相关协议顺序**：



其中，CreateRole不是必须的，只有在GetRoleInfo\_Re返回错误“NO\_ROLE”时，才会触发CreateRole，错误码在scripts/common.lua中定义，所有游戏协议的retcode都会使用这些错误码。