# 客户端网络模块说明文档

## 需求定义

客户端网络模块需要实现的功能主要有两个方面：

1. 使用class ntcp\_client和服务器通信；
2. 将Native网络层封装到.NET层，实现Managed层的网络消息的收发。

## 依赖关系



图：客户端网络模块NexusNetworkCLI依赖关系图

## 程序实现

## 总体结构



### 继承ntcp\_client

完成和服务器的通信使用class ntcp\_client即可，但是需要将网络层（Socket的封装）和客户端消息收发结合起来还需要将class ntcp\_client进一步扩展，class client\_connection: public nexus::ntcp\_client用来作为网络层和逻辑层结合的中间层，主要实现以下件事：

1. 记录连接状态；
2. 接受消息的消息队列；
3. 消息监听器，监听器的目的主要用来将Native Message发送到Managed code；

同时public ref class NClientConnection完成了将class client\_connection封装到托管层

### 消息监听器（msg\_listener）

监听器的设计目的主要用来将Native Message发送到Managed code，这里是一个典型的Native code访问managed code的例子。



图：消息监听器，一种典型的Native code调用Managed code的方法

### 消息枚举定义

Native message消息ID并不能再managed code层使用，需要在托管代码层重定义这些消息ID，同时需要保证跨语言后消息ID还能对应上，因此最好的重定义这些消息ID的地方就是CLI层，在ProtocolEnum.h文件中对消息ID进行重定义，如：

public enum class NetLogin : System::UInt16

{

CS\_LoginProof = nexus::C2S\_LoginProof,

SC\_LoginProofResult = nexus::S2C\_LoginProofResult,

};

### 消息协议定义、Structure IntPtr之间的互相转换

Native message structure在C#中不能直接访问，因此需要有一种办法能够将Native message structure变成C#可以自由访问的结构体。

这里需要用到System.Runtime.InteropServices.Marshal库, 它提供了一个方法集，这些方法用于分配非托管内存、复制非托管内存块、将托管类型转换为非托管类型，此外还提供了在与非托管代码交互时使用的其他杂项方法。

internal static void SendNetMessage<T>(T message)

{

**int messageSize = Marshal.SizeOf(message);**

**// Initialize unmanged memory to hold the struct.**

**IntPtr rawDataPtr = Marshal.AllocHGlobal(messageSize);**

try

{

**// Copy the struct to unmanaged memory.**

**Marshal.StructureToPtr(message, rawDataPtr, false);**

// send network message

unsafe

{

**Program.game.ClientNetConnection.SendMessage((void\*)rawDataPtr, (UInt32)messageSize);**

}

}

finally

{

**// Free the unmanaged memory.**

**Marshal.FreeHGlobal(rawDataPtr);**

}

}

public static T PtrToStruct<T>(IntPtr rawDataPtr) where T : struct

{

T result = default(T);

**result = (T)Marshal.PtrToStructure(rawDataPtr, typeof(T));**

return result;

}

这里需要注意几个地方：

1. 重定义一份和Native消息结构内存大小相同的结构体，如：

// 请求登陆验证

CMD\_START(C2S\_LoginProof)

CMD\_END

---------------------------------------------------------------------------------

/// <summary>

/// 请求登陆验证

/// </summary>

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, **Pack = 1**)]

public struct C2S\_LoginProof

{

public const UInt16 Id = (UInt16)NetLogin.CS\_LoginProof;

public UInt16 messageId;

}

1. 发送消息的时候使用SendNetMessage<T>(T message)泛型函数，它完成了申请非托管内存，copy数据到非托管内存两件很重要的事情；
2. 接受消息解析时，我们用了一个技巧那就是在Native code层就已经将消息的ＩＤ解析好了抛到managed code层，因此我们知道ID的情况下就知道了应该使用哪个结构体去解析数据了，如：

if(e.messageId == (UInt16)NetLogin.SC\_LoginProofResult)

{

**S2C\_LoginProofResult ret = MarshalConversion.PtrToStruct<S2C\_LoginProofResult>(e.MessagePtr);**

}

### 消息派发

前面的内容都是在绕圈子，如何将在不同语言之间完成内存访问等，接下来的任务就是如何将接受的网络消息派发到不同的游戏逻辑模块去使用。

派发的方式时使用C# event-delegate机制：

// 初始化网络设置

ClientNetMessageListener messageListener = new ClientNetMessageListener();

messageListener.NetConnectAborted += new EventHandler<NetMessageEvent>(messageListener\_NetConnectAborted);

messageListener.NetConnected += new EventHandler<NetMessageEvent>(messageListener\_NetConnected);

messageListener.NetConnectError += new EventHandler<NetMessageEvent>(messageListener\_NetConnectError);

messageListener.NetConnectTimeout += new EventHandler<NetMessageEvent>(messageListener\_NetConnectTimeout);

messageListener.NetDisconnected += new EventHandler<NetMessageEvent>(messageListener\_NetDisconnected);

messageListener.NetMessageReceived += new EventHandler<MessageReceivedEvent>(messageListener\_NetMessageReceived);

clientNetConnection.AddListener(messageListener);

clientNetConnection.Startup(20, true);

## 小结

客户端网络模块封装比较到位，逻辑层需要注意的几点：

1. 定义消息ID，managed code的枚举和native code的枚举值必须对应；
2. 定义和Native message structure内存大小一致的结构体；
3. 使用泛型函数发送和解析消息；

目前已知的缺陷有：

1. 变长消息的解析，如何处理变长消息的解析；
2. 如何处理union ;
3. 如何保证Native code层和managed code层同步修改。