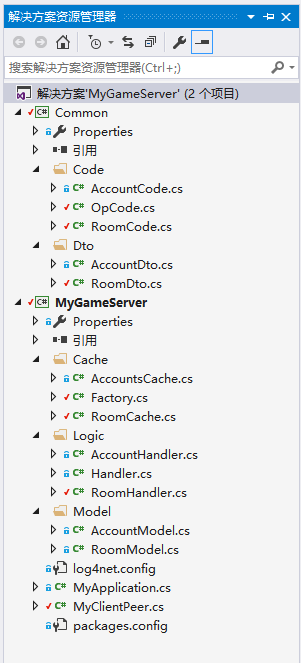
# **Unity 使用Photon Server 创建一个简单聊天室**

参考教程：[基于PhotonServer单服聊天室](http://www.taikr.com/course/499" \t "https://blog.csdn.net/l773575310/article/details/_blank) http://www.taikr.com/course/499  
完整Github工程（服务器）：[MyGameServer](https://github.com/ZeroChiLi/MyGameServer" \t "https://blog.csdn.net/l773575310/article/details/_blank) https://github.com/ZeroChiLi/MyGameServer  
完整Github工程（Unity客户端）：[PhotonChatRoom](https://github.com/ZeroChiLi/PhotonChatRoom" \t "https://blog.csdn.net/l773575310/article/details/_blank) https://github.com/ZeroChiLi/PhotonChatRoom  
数据库用户列表操作见另一篇：[Unity C# 连接SQL Server数据库，实现获取和添加登录注册的用户列表](http://blog.csdn.net/l773575310/article/details/72916526" \t "https://blog.csdn.net/l773575310/article/details/_blank)https://blog.csdn.net/l773575310/article/details/72916526

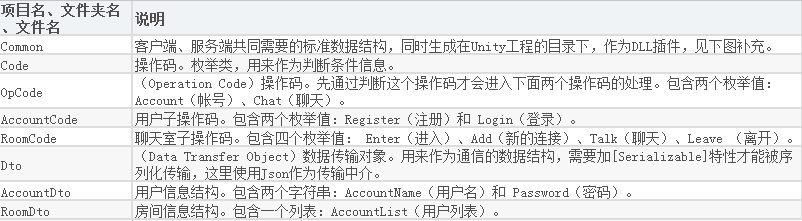
　　初步了解使用Photon插件如何进行通信，按照上面链接的教程走了一遍，掌握了一些简单操作。

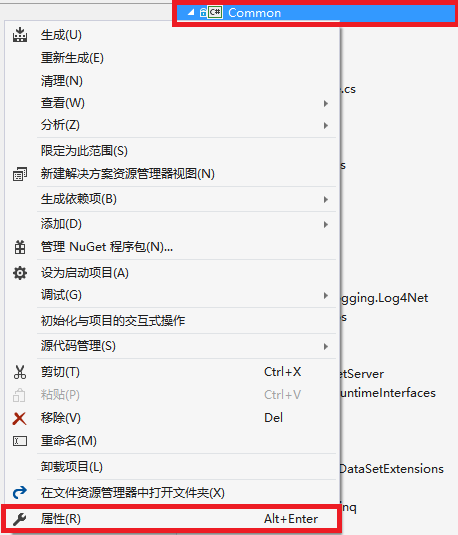
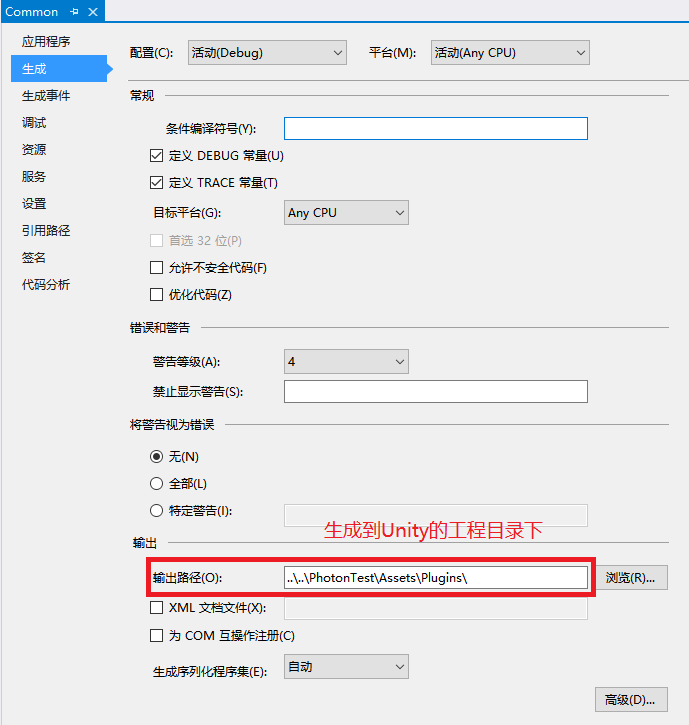
# **MyGameServer服务器配置**

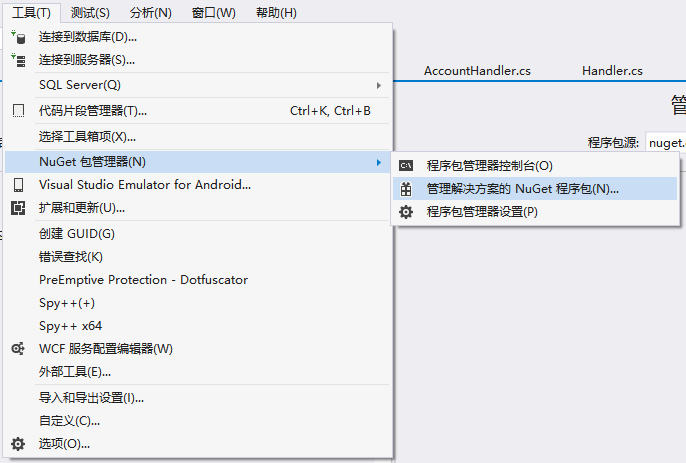
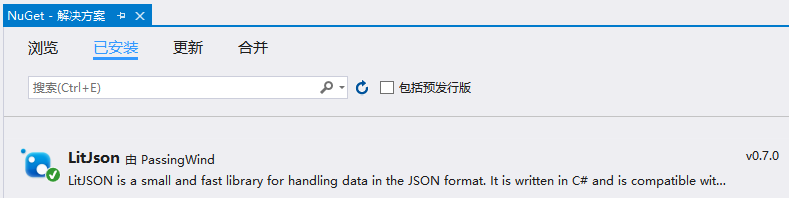
## **工程结构**



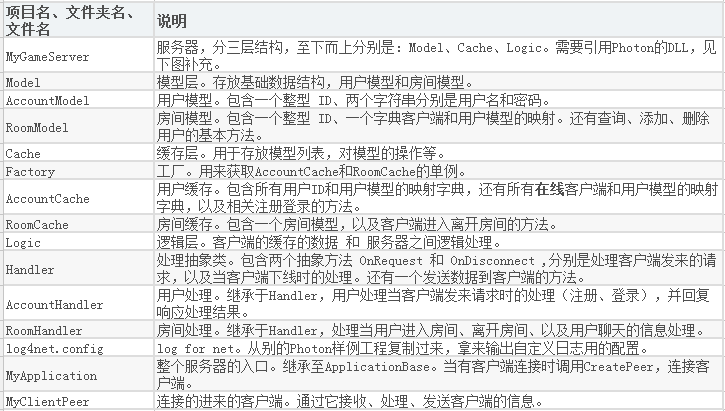
**客户端、服务器规定通用操作码与数据结构**

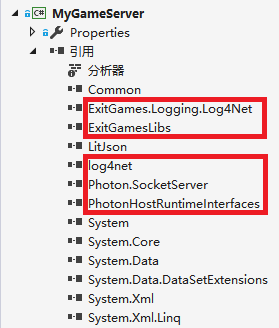


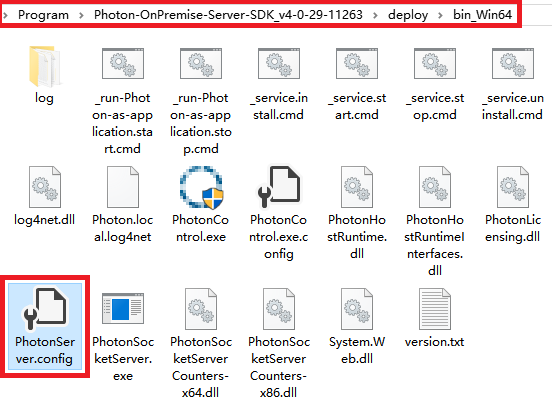
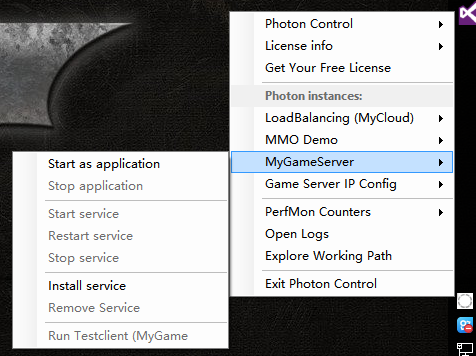
****将Common生成到Unity工程目录下。方便Untiy直接调用。****   
 

**传输Dto时需要转换成Json，添加LitJson插件。   
 **

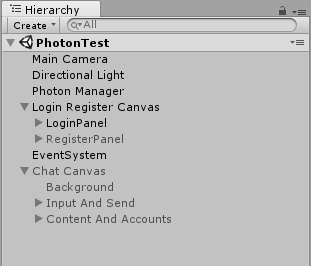
**服务器结构**



****服务器所需要添加的引用，添加在服务器工程lib目录下，从Photon的lib目录下找到下面这些DLL。****   


****配置到Photon，便于启动服务器。****   
[配置文件点我：PhotonServer.config](https://github.com/ZeroChiLi/MyGameServer/tree/master/Photon_deploy_bin_Win_Config" \t "https://blog.csdn.net/l773575310/article/details/_blank) https://github.com/ZeroChiLi/MyGameServer/tree/master/Photon\_deploy\_bin\_Win  
 

**客户端结构**



**UI介绍**

|  |  |
| --- | --- |
| Photon Manager--。 | 包含PhotonManager类，用于处理客户端与服务端通信的桥梁 |
| LoginPanel--登录UI界面。两个按钮分别是登录到聊天房间，以及进入注册界面。 | IMG_265 |
| RegisterPanel--注册UI界面。用于注册帐号。 | IMG_266 |
| ChatCanvas--聊天UI界面。上部分分别是显示聊天内容，以及用户列表。下部分分别是聊天信息输入和发送信息。 | IMG_267 |

# **客户端脚本介绍**

**PhotonManager**

　　Photon客户端管理。实现了IPhotonPeerListener接口。

// PhotonManager

private void Awake() {

instance = this; //客户端单例

peer = new PhotonPeer(this, protocol); //创建客户端，协议是UDP

DontDestroyOnLoad(accountReceiver);

}

private void Update(){

if (!connected) peer.Connect(serverAddress, applicationName); //连接服务器

if(Time.time > currentTime) {

peer.Service(); //获取服务，并不需要每帧获取，简单优化

currentTime += updateIntervalTime;

}

}

private void OnDestroy(){

peer.Disconnect(); //断开连接

}

//向服务器发请求，参数：操作码，参数数据字典，子操作码

public void OnOperationRequest(byte opCode, Dictionary<byte, object> parameters = null, byte SubCode = 0) {

//规定键'80'对应的是子操作码

parameters[80] = SubCode;

peer.OpCustom(opCode, parameters, true);

}

//服务端发送过来的响应

public void OnOperationResponse(OperationResponse response){

Debug.Log(response.ToStringFull());

byte subCode = (byte)response.Parameters[80];

switch ((OpCode)response.OperationCode) { //判断发过来操作码，做出响应。

case OpCode.Account: accountReceiver.OnReceive(subCode, response);

break;

case OpCode.Room: chatRceiver.OnReceive(subCode,response);

break;

}

}

**IReceiver**

　　接收响应接口。

interface IReceiver{

//接受服务器响应，参数：子操作码，响应信息。

void OnReceive(byte subCode, OperationResponse response);

}

**AccountReceiver**

　　接收操作用户登录注册的信息响应。实现 IReceiver接口。

public AccountView accountView;

public void OnReceive(byte subCode, OperationResponse response) {

switch ((AccountCode)subCode) {

case AccountCode.Register:

if (response.ReturnCode == 0) { //返回码正确时

accountView.OnHideRegisterPanel(); //关闭注册面板。

break;

case AccountCode.Login:

if (response.ReturnCode == 0) { //返回码正确时

//告诉服务器可以将进入房间了

PhotonManager.Instance.OnOperationRequest((byte)OpCode.Room, new Dictionary<byte, object>(), (byte)RoomCode.Enter);

}

break;

}

}

**ChatReceiver**

　　接收操作聊天房间的信息响应。实现 IReceiver接口。

public void OnReceive(byte subCode, OperationResponse response) {

switch ((RoomCode)subCode) {

case RoomCode.Enter: //本人进入房间处理

if (response.ReturnCode == 0) {

loginCanvas.SetActive(false); //隐藏登录面板

chatCanvas.SetActive(true); //显示聊天面板

//获取房间信息并初始化

chatView.Init(GetResponseFromJson<RoomDto>(response));

}

break;

case RoomCode.Add: //房间有新用户处理

chatView.AddAccount(GetResponseFromJson<AccountDto>(response));

break;

case RoomCode.Talk: //房间有人说话处理

string text = response.Parameters[0].ToString();

chatView.Append(text);

break;

case RoomCode.Leave: //房间有人离开处理

chatView.LeaveRoom(GetResponseFromJson<AccountDto>(response));

break;

default:

break;

}

}

//从获取到的Json信息中提取出Dto

private Dto GetResponseFromJson<Dto>(OperationResponse response) {

return JsonUtility.FromJson<Dto>(response.Parameters[0].ToString());

}

**AccountView**

　　用户登录和注册界面UI响应。

//登录和注册发送用户信息，子操作码分别是：AccountCode.Login和AccountCode.Register

public void SendAccountRequest(string accountName,string password,byte subCode) {

AccountDto dto = new AccountDto();

dto.AccountName = accountName;

dto.Password = password;

Dictionary<byte, object> parameters = new Dictionary<byte, object>();

parameters[0] = JsonUtility.ToJson(dto);

PhotonManager.Instance.OnOperationRequest((byte)OpCode.Account, parameters, subCode);

}

**ChatView**

　　聊天房间界面UI响应，包括一个用户名（字符串）和对应Perfab的字典。用户的进入和离开都操作这个字典。

# **测试结果**

|  |  |
| --- | --- |
| **1. 运行测试。** | IMG_268 |
| **2. 登录一个没注册的号，登录失败。** | IMG_269 |
| **3. 注册一个充满优秀品质的帐号。** | IMG_270 |
| **4. 登录该账户并发送一条惊人的信息。** | IMG_271 |
| **5. 在另一个客户端登录另一个帐号，并发送信息。** | IMG_272 |
| **6. 回到原来的客户端查看。** | IMG_273 |

数据库用户列表操作见另一篇：[Unity C# 连接SQL Server数据库，实现获取和添加登录注册的用户列表](http://blog.csdn.net/l773575310/article/details/72916526" \t "https://blog.csdn.net/l773575310/article/details/_blank)https://blog.csdn.net/l773575310/article/details/72916526